HUMAN INTERFACE DESIGN

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS

Dr. Dobb's Journal

#102 April 1985

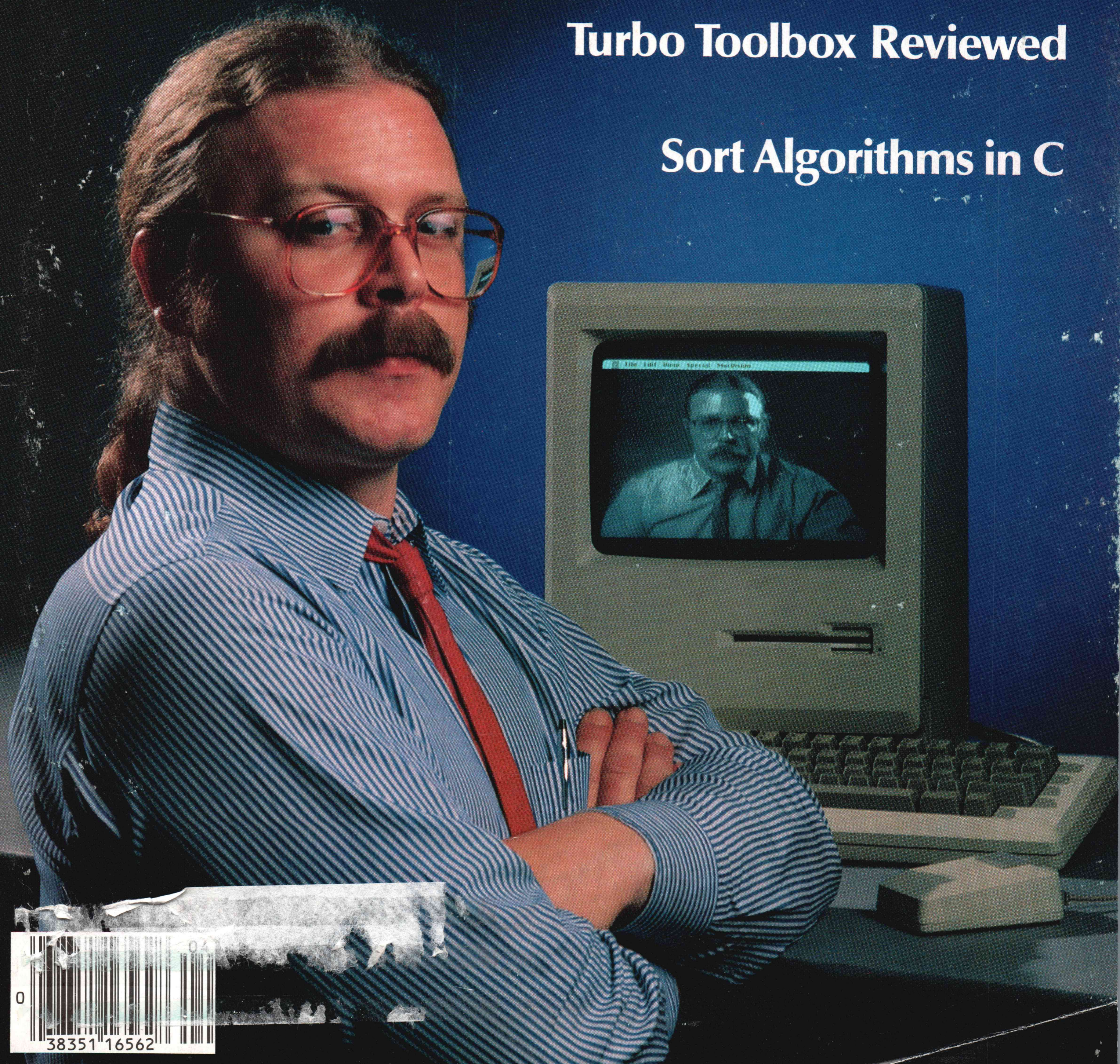
$2.95 (3.50 Canada)

80286 Xenix

Three Printer-taming Programs

Turbo Toolbox Reviewed

Sort Algorithms in C



Software Development Tools

If:

Manufacturer-Compatible, Fully-Supported Cost-Effective, Cross and Native Development tools are important to you!

Then:

Let us tell you about MICROTEC RESEARCH's extensive line of field-proven software development tools for serious software developers.

C
Pascal
Assemblers
Simulators

Manufacturer Compatible

MICROTEC RESEARCH has been providing flexible and economical solutions for software developers since 1974. We've grown with the industry as our cross software development tools have transformed many large and small computers into powerful cross development systems for microprocessor applications. Our software tools are manufacturer compatible. Therefore, software made using manufacturers development systems may be more productively used in the MICROTEC RESEARCH cross-development environment.

Fully-Supported

MICROTEC RESEARCH's products are continually maintained and updated based upon the experience of thousands of installations worldwide. Revisions and modifications are distributed to licensees in warranty or covered by a service contract. Upgrades, which are major enhancements to a product's capabilities, are available at a significant discount. Our update/upgrade policy assures users they'll always be current with the fast moving world of microprocessor development.

Effective Differences

Beginning with product concept, through development, quality assurance, and post-sales support — **Quality, Compatibility,** and **Service** are the differences which set MICROTEC RESEARCH apart from the others.

Start Saving Time & Money

If you are a serious software developer, shopping for software development tools, call or write today for more information. **800-551-5554** In CA call **(408) 733-2919**

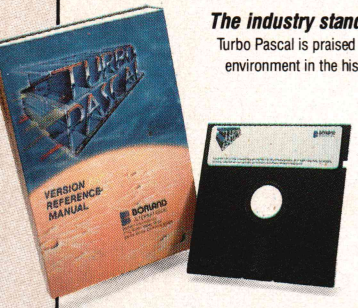
Target Microprocessors	Host Mini's	Host PC's	Software Tools
8086/80186	DEC VAX, PDP-11	IBM PC	C Compilers
8096	DG MV-Series Eclipse	Data General/ONE	Pascal Compilers
8080/8085	Apollo	HP 150	Assemblers
8051	UNIX Systems	COMPAQ	Simulators
8048	others	Columbia	Linking Loaders
others		DEC Rainbow	Librarians
		Other Compatibles	Conversion/ Download
			Communications
			VT-100 Emulator

for Serious Software Developers

MICROTEC™ RESEARCH

3930 Freedom Circle, Suite 101, Santa Clara, CA 95054
Mailing Address: P.O. Box 60337, Sunnyvale, CA 95088
(408) 733-2919 Telex (ITT) 4990808

Speed, Power, Price.
Borland's Turbo Pascal Family.



The industry standard. With more than 250,000 users worldwide Turbo Pascal is the industry's de facto standard. Turbo Pascal is praised by more engineers, hobbyists, students and professional programmers than any other development environment in the history of microcomputing. And yet, Turbo Pascal is simple and fun to use!

$69.95

Jeff Duntemann, PC Magazine: *"Language deal of the century . . . Turbo Pascal: It introduces a new programming environment and runs like magic."*

Dave Garland, Popular Computing: *"Most Pascal compilers barely fit on a disk, but Turbo Pascal packs an editor, compiler, linker, and run-time library into just 29K bytes of random-access memory."*

Jerry Pournelle, BYTE: *"What I think the computer industry is headed for: well documented, standard, plenty of good features, and a reasonable price."*

Portability. Turbo Pascal is available today for most computers running PC DOS, MS DOS, CP/M 80 or CP/M 86. A XENIX verison of Turbo Pascal will soon be announced, and before the end of the year, Turbo Pascal will be running on most 68000 based microcomputers.

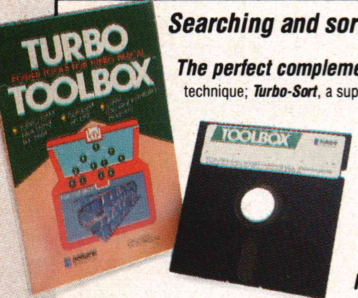
High resolution monochrome graphics for the IBM PC and the Zenith 100 computers

Dazzling graphics and painless windows. The Turbo Graphix Toolbox will give even a beginning programmer the expert's edge. It's a complete library of Pascal procedures that include:

—Full graphics window management.
—Tools that will allow you to draw and hatch pie charts, bar charts, circles, rectangles and a full range of geometric shapes.
—Procedures that will save and restore graphic images to and from disk.
—Functions that will allow you to precisely plot curves.
—Tools that will allow you to create animation or solve those difficult curve fitting problems.
and much, much more . . . . .

$54.95
NEW

No sweat and no royalties. You may incorporate part, or all of these tools in your programs, and yet, we won't charge you any royalties. Best of all, these functions and procedures come complete with commented source code on disk ready to compile!



Searching and sorting made simple

The perfect complement to Turbo Pascal. It contains: ***Turbo-Access***, a powerful implementation of the state-of-the-art B+tree ISAM technique; ***Turbo-Sort***, a super efficient implementation of the fastest data sorting algorithm, "Quicksort on disk". And much more.

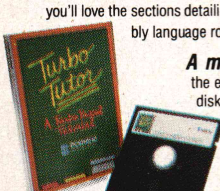
$54.95

Jerry Pournelle, BYTE: "The tools include a B+tree search and a sorting system; I've seen stuff like this, but not as well thought out, sell for hundreds of dollars."

Get started right away: free database! Included on every Toolbox disk is the source code to a working data base which demonstrates how powerful and easy to use the Turbo-Access system really is. Modify it to suit your individual needs or just compile it and run.

Remember, no royalties!

From Start to Finish in 300 pages. Turbo Tutor is for everyone, from novice to expert. Even if you've never programmed before, Turbo Tutor will get you started right away. If you already have some experience with Pascal or another programming language, Turbo Tutor will take you step by step through topics like data structures and pointers. If you're an expert, you'll love the sections detailing subjects such as "how to use assembly language routines with your Turbo Pascal programs."



A must. You'll find the source code for all the examples in the book on the accompanying disk ready to compile. Turbo Tutor might be the only reference on Pascal and programming you'll ever need.

$34.95

NOT COPY-PROTECTED

TURBO PASCAL FAMILY

Available at better dealers nationwide. Call (800) 556-2283 for the dealer nearest you. To order by Credit Card call (800) 255-8008, CA (800) 742-1133

Pascal 3.0	$ 69.95	____
Pascal w/8087	$109.90	____
Pascal w/BCD	$109.90	____
Pascal w/8087 & BCD	$124.95	____
Turbo Toolbox	$ 54.95	____
Turbo Graphics	$ 54.95	____
Turbo Tutor	$ 34.95	____

*These prices include shipping to all U.S. cities. All foreign orders add $10.

Carefully Describe your Computer System!

Mine is: ___ 8 bit ___ 16 bit
I Use: ___ PC-DOS ___ MS-DOS ___ CP/M 80 ___ CP/M 86

My computers' name/model is: ____________

The disk size I use is:
☐ 5¼" ☐ 8" ☐ 3½"

Amount: (CA 6% tax) ____________

Payment: VISA MC BankDraft Check

Credit Card Expir. Date: ____/____

Name on Card: ____________

Card #: ____________

Name: ____________

Shipping Address: ____________

City: ____________

State: ________ Zip: ________

Telephone: ____________

F11

COD's and Purchase Orders WILL NOT be accepted by Borland. California residents: add 6% sales tax. Outside USA: add $10 and make payment by bank draft, payable in US dollars drawn on a US bank.

BORLAND INTERNATIONAL
Software's Newest Direction
4113 Scotts Valley Drive
Scotts Valley, California 95066
TELEX: 172373

Circle **no. 11** on reader service card.

Up Your AT™ for $56 per Megabyte!

■ **While our specifications certainly speak for themselves, we thought you still might like to hear from some of our users:**

■ "Emerald Systems expands the potential of PCs by providing the ability to access large amounts of data on line, quickly and reliably."
Terry Baptiste, Computerland, Lafayette, Ca.

■ "Service and support is great, which is an unusual experience. Emerald's software for backup and restore is invaluable. Can't put a price on it. Productivity and efficiency has increased at least 50%!"
Bruce Kittinger, Pinon Systems, Ft. Collins, Co.

■ Runs like a champ with 3-Com Ethernet."
Alvaro Ramirez, Micro Age, Miami, Fl.

■ ". . . high capacity and flexibility. At last, a tape back up we can count on. And the price is right!"
John Acres, EDT, Las Vegas, Nv.

■ "The speed at which you can back up is very impressive."
Jim McEwen, Mercy Hospital, Portland, Me.

■ "When Emerald says your unit will be there on Thursday, it's there on Thursday! Delighted we were able to exceed the usual 32 megabyte restriction."
Steven Mayer, Take One Company, New York, N.Y.

■ "The Emerald 70 MB hard disk is extremely easy to install and work with. Emerald has a complicated piece of equipment made easy to use."
Tom Edler, Jewish Hospital, St. Louis, Mo.

■ "Our Emerald fixed disk installed quickly and easily. Emerald's reliable disk and tape backup further enhances LIBRA's high function accounting software."
Kenn White, Libra Programming, Salt Lake City, Ut.

HARD DISK

We've broken through the 32 MByte DOS barrier!

Now you can create up to 240 MByte databases on one file, with multiple volumes per disk drive. You get 14 times more storage with a 30% increase in access speed!

And that's not all:

Expands up to 280 MB for as little as $56 per megabyte.
6 expansion slots
Up to 24 volumes
Drive sizes: 40,70,140
Internal or External
Also for PC,XT, and compatibles
Supports all PC compatible networks
Simple menu-driven installation

TAPE BACKUP

¼" cartridge
½" 9 track
60 Megabytes
No lost data from tape run out
Backup and Restore Utility (BRU™) software included
LAN Compatible*

***FREE APPLICATION GUIDE**: "IBM LAN Installation and Implementation."

Call (619) 270-1994, or write to

Emerald™ Systems Corporation
Mainframe Storage for Micros
4901 Morena Blvd,
San Diego, 92117
TLX 323458 EMERSYS
EASYLINK 62853804

Distributed by Manchester Equipment of NYC, and selected Entre, MicroAge and Computerland stores.

EMERALD™
SYSTEMS CORPORATION
Mainframe Storage for Micros™

Circle **no. 114** on reader service card.

Circle **no. 32** on reader service card.

Dr. Dobb's Journal

Editorial
Editor-in-Chief *Michael Swaine*
Editor *Randy Sutherland*
Managing Editor *Frank DeRose*
Technical Editor *Alex Ragen*
Editorial Assistant *Sara Noah*
Contributing Editors *Robert Blum, Dave Cortesi, Ray Duncan, Allen Holub*
Copy Editor *Polly Koch*
Typesetter *Jean Aring*

Production
Design/Production Director *Detta Penna*
Art Director *Shelley Rae Doeden*
Production Assistant *Alida Hinton*
Cover *Tom Upton*

Advertising
Advertising Director *Stephen Friedman*
Advertising Sales *Walter Andrzejewski, Shawn Horst, Beth Dudas, Michele Beaty*
DDJ Classifieds *Alice Abrams*
Advertising Coordinators *Lisa Boudreau, Jay Horvath*

Circulation
Fulfillment Manager *Stephanie Barber*
Direct Response Coordinator *Maureen Snee*
Promotions Coordinator *Jane Sharninghouse*
Circulation Assistant *Kathleen Boyd*

Administration
Finance Manager *Sandra Dunie*
Business Manager *Betty Trickett*
Accounting Coordinator *Mayda Lopez-Quintana*
Accounts Payable *Denise Giannini*
Billing Coordinator *Laura Di Lazzaro*

M&T Publishing, Inc.

Chairman of the Board *Otmar Weber*
Director *C.F. von Quadt*
President *Laird Foshay*

Dr. Dobb's Journal (USPS 307690) is published monthly by M&T Publishing, Inc., 2464 Embarcadero Way, Palo Alto, CA 94303, (415) 424-0600. Second class postage paid at Palo Alto and at additional entry points.

Address correction requested. Postmaster: Send Form 3579 to *Dr. Dobb's Journal*, 2464 Embarcadero Way, Palo Alto, CA 94303. **ISSN 0278-6508**

Subscription Rates: $25 per year within the United States, $44 for first class to Canada and Mexico, $62 for airmail to other countries. Payment must be in U.S. Dollars, drawn on a U.S. Bank. For subscription problems, call: outside of CA 1-800-321-3333; within CA 1-619-485-6535.

Foreign Distributors: ASCII Publishing, Inc. (Japan), Computer Services (Australia), Computer Store (New Zealand), Computercollectief (Nederland), Homecomputer Vertriebs GMBH (West Germany), International Presse (West Germany), La Nacelle Bookstore (France), McGill's News Agency PTY LTD (Australia), Progresco (France).

People's Computer Company

Dr. Dobb's Journal is published by M&T Publishing, Inc. under license from People's Computer Company, 2682 Bishop Dr., Suite 107, San Ramon, CA 94583, a non-profit, educational corporation.

April 1985
Volume 10, Issue 4

CONTENTS

Dr. Dobb's Board of Referees

Cover Credit

Tom Upton created our cover this month with the help of MacVision, a graphic digitizer from Koala Technologies Corporation.

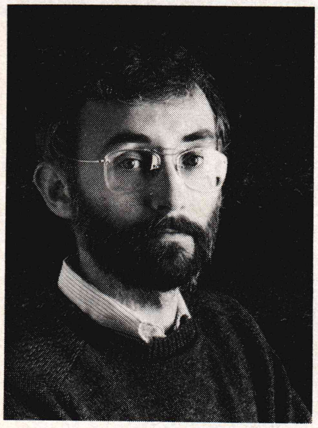
SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS

Dr. Dobb's Journal

ARTICLES

COLUMNS

DEPARTMENTS



Changes come and trip all your traps. You subscribe to *Micro* and you get *Dr. Dobb's Journal*. You just get used to thinking of yourself as a "serious computerist" and you find yourself a subscriber to a magazine "for advanced programmers."

With this issue, we welcome several thousand readers of the sadly-defunct *Micro* magazine to *Dr. Dobb's. Micro*, like *Microsystems* and *Microcomputing*, died last fall in a bad season for many technical computer magazines.

I'm sorry about *Micro*, but I'm happy to see this influx of readers serious about the 6502 and 68000 microprocessors. We've been publishing more and more 68000 material recently, but aside from an occasional Commodore 64 piece, we've neglected the 6502 programmer. The *Micro* readers present *Dr. Dobb's* with an opportunity to address an audience it's lost touch with in recent years. (For an indication of the kind of 6502 material we have published in the past, see the note at the end of the "Realizable Fantasies" column elsewhere in this issue.)

Welcome, *Micro* readers.

Changes have come to *Dr. Dobb's* as well in the past year. This is the twelfth issue of the magazine published under the *M&T* banner (the hundred-and-second in all). I became editor-in-chief then (and thanks, by the way, to all those I failed to thank in my February editorial), and I instituted some new features in the magazine, including our first operating-systems issue (on Unix) and our unconventional coverage of the Mac (see our January how-to on home-fattening your Mac or this issue's Realizable Fantasy on liberating the Mac).

While making changes, I've tried not to lose track of what *Dr. Dobb's* was founded to do: to provide software tools for advanced programmers. In 1976, those programmers were hackers and in 1985 they tend to be professional software designers, but after attending last fall's Hackers' Conference I've decided that the distinction is moot.

Changes come thick and fast in the microcomputer industry, but in the thick of it all, some things must hold fast. Without *Micro*, without *Microsystems*, without *Microcomputing*, where does the programmer interested in 8-bit systems go? We promised, in our May and October issues, not to abandon the 8-bit computer, and we haven't. We promised not to forsake public-domain and low-cost software, and we haven't. (We've also pointed out that purveyors of public-domain and commercial software must coexist.)

But there are changes yet to come. While we will not abandon our current readers, we'll be exploring new territory in 1985, ranging beyond the fields we know. The only way we can do this is to publish more material that is of broader applicability—not by lowering the technical level, but by giving more emphasis to algorithms, to techniques, to portable software, and less to machine-specific code. I claim that the overall effect will be to raise, not lower the technical level of the magazine without making it harder to understand. We'll see if I'm right.

We'll also give more attention to the craft of software design in 1985, and to the aesthetics of design. If writing utilities produces utilitarian code, then I think that programmers should be writing efficiencies, elegances, beauties in addition to utilities, because we need more efficient, more elegant, more beautiful software. We'll encourage that.

And yes, I deliberately worked into this editorial the themes of my other eleven editorials. I'm sorry; I learned recursion at an early age and was warped for life.

Michael Swaine

Michael Swaine

Chubby Mac

Dear *DDJ*,
I would like to congratulate you for the excellent January [1985, #99] issue of *Dr. Dobb's* featuring "Fatten Your Mac" by Tom Lafleur and Susan Raab. Thanks to that article, I was able to upgrade my Mac for much less than Apple's list price: I paid $200 total or around $11 per memory chip.

Having never seen a hardware technical manual for the Mac that's equivalent to "Inside Macintosh" for software, I found the diagnostics part of the article very useful. While doing the upgrade, I encountered socket connection problems that were solved by following the continuity test procedure mentioned.

Do you know what dealers or customers do with the 128K boards, for those who upgrade their boards the official Apple way? I was thinking that if I decide to pursue this and upgrade Macs for friends and for money, I will probably need some extra 128K boards as a safety net. If you know people who have a 128K board that they are going to throw away, let me know.

I am so happy to have a 512K Mac now that I decided to enter a 1-year subscription to *Dr. Dobb's*. Once again, congratulations to *Dr. Dobb's* and the authors.

Sincerely,
Tho AuTruong
908-B Menlo Avenue
Menlo Park, CA 94025

PUBlic Service

Dear *DDJ*,
A minor correction is needed to our article "CP/M 2.2 Goes Public (November, 1984, #97)". In Listing 2, page 70, the PUBLIC.ASM utility to set or reset the PUBlic file attribute bit contains a bug, discovered by Greg Platt, that prevents it from making very large files PUBlic.

Here's the fix; Update the version equate to read:

```
vers    equ     1$1
```

and insert the following code in the SAMEXT routine preceeding its "ret" instruction:

```
rnz          ; v 1.1
inx     h    ;extent is 0,
               check overflow (s2)
               ext.
inx     h
mov     a,m
ani     7fh
;
ret          ;end of SAMEXT
               routine
```

PUBPATCH—the BDOS patch that implements PUBlic file processing—is unaffected by the bug.

Sincerely,
Bridger Mitchell
Derek McKay
Plu*Perfect Systems
Box 1494
Idyllwild, CA 92349

Dear *DDJ*,
The article "CP/M 2.2 Goes PUBlic" by Mitchell and McKay, in your November [1984, #97] issue was most informative and useful. However, the code text published with it had a few problems. Some specifics from Listing One: the location definitions CKFILPOS and SETSTAT were multiply defined (to be sure, the definitions were the same, but still rather irregular); 'nxbyt:' was referenced both as itself and 'nxtbyte', typo or program error I can't tell, since only the first six characters are frequently used. More serious was the use of a font in which both lower case 'L' and the number one used the same representation. This is common on typewriters, but rare on computer printers. Line 253 is definitely ambiguous as printed, although it can be deciphered in context.

At any rate, I was able to enter the program text, make the necessary corrections, translate it to true Z80 code, and assemble it with the Z80MR assembler from MicroCornucopia. Installation under ZCPR3 required a little research to find the correct addresses, but they were there. Thus I no longer need to have long text files in the same user area as my PW or WS editor; this is a real convenience in the rather crowded directories on my Kaypro10. My sincere thanks to the authors!

Gratefully
Bonnell Frost
5559 Colt Drive
Longmont, CO 80501

Grep Again

Dear *DDJ*,
A belated thank you for grep.c [*DDJ* #96, October 1984], which I finally got around to typing in and compiling on my Osborne 1 with C/80.

C/80 Version 3.0 from The Software Toolworks will not compile grep.c as listed because it has no:

a) linker
b) typedef statement
c) FILE type
d) #defined NULL
e) stdin
f) stdout
g) stderr
h) fgets
i) stdio.h
j) capability to call a function with a different number of

arguments than defined.

To compile grep.c with C/80:

In TOOLS.H:

1. After #define NUL 0,
   add #define NULL NUL
2. Add #define EOF –1
3. Add #define TOKEN struct
   token
4. As explained in Section 8 of the C/80 Version 3.0 documentation add the following:
   #define FILE int
   #define stdin fin
   #define stdout fout
   #define stderr 0
   extern int fin, fout;
5. In the typedef statement on page 58 delete the words "typedef" and "TOKEN".

In TOOLS.C:

1. Delete the two #include directives
2. On page 61 add the argument "boln" to the omatch( ) call. The modified line should read:
   while(*lin && omatch(&lin,
   pat, boln))
3. On page 64 change the expression
   (dstart – dest < maxccl)
   to
   (dest – dstart < maxccl)

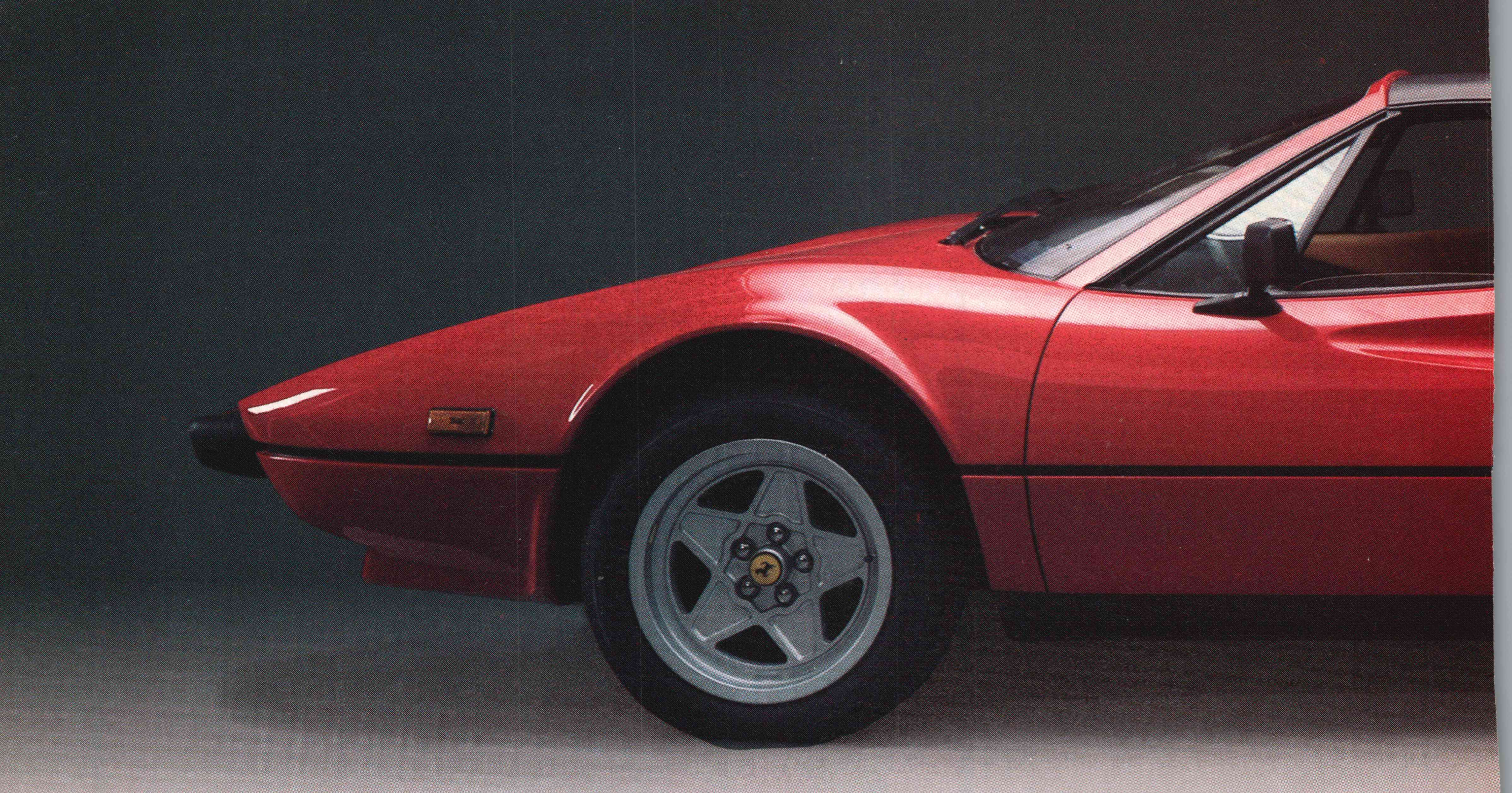
In GREP.C:

1. Delete the line #include "a:stdio.h"
2. Change #include "b:tools.h" to #include "tools.h"
3. Add #include "printf.c"
4. Add #include "tools.c"
5. On page 77 change
   if(matchs(line, exprv[j]))
   to
   if(matchs(line, exprv[j],0))
6. On page 83 change
   bdos(11)
   to
   bdos(11,0)
7. At the end of grep.c add #include "STDLIB.C"

Thank you,

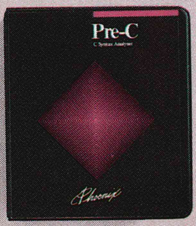
Robert C. Briggs
1337 Rossway Court
Los Altos, CA 94022

DDJ



Programmers' Pfantas

Phoenix makes programmers' dreams come true. With the best-engineered, highest performance programming tools you can find. A full line of MS™-DOS/PC DOS programs and utilities no other company offers. All designed to help you write, test and deliver the best programs possible. Top-of-the-line quality at a price you can afford.



Finally, A Lint For MS-DOS.

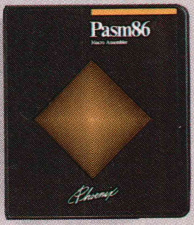
Now you can get the full range of features C programmers working in UNIX™ have come to expect from their Lint program analyzer.

With Pre-C™ you can detect structural errors in C programs five times faster than you can with a debugger. Find usage errors almost impossible to detect with a compiler. Cross-check multiple source files and parameters passed to functions. Uncover interface bugs that are difficult to isolate. All in a single pass. Capabilities no C compiler, with or without program analyzing utilities, can offer. In fact, Pre-C outlints Lint, since you can handle analyses incrementally.

And, Pre-C's flexible library approach lets you maintain continuity across all the programs in your shop, whether you use Pre-C's pre-built libraries, pre-existing functions you already have, or some you might want to buy yourself.

Plus, you're not limited to one particular library, and Pre-C keeps track of all the libraries you're using to make sure that code correctly calls them. **$395.**

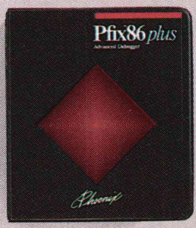
Special Thanks to Gasten Audrey of Framingham, MA.



Assemble Programs Twice As Fast.

Pasm™86 assembles Masm files 2 to 3 times faster than Masm 3.0. Pasm86 supports 8086/88, 8087, 80186 and 80286 processors.

With Pasm 86's built-in defaults, you can write code quickly since you won't spend hours learning all the control statements needed at the beginning of your program. You can define symbols on the command line. Decide whether you want error messages or not. And, put local symbols within procedures. **$295.**

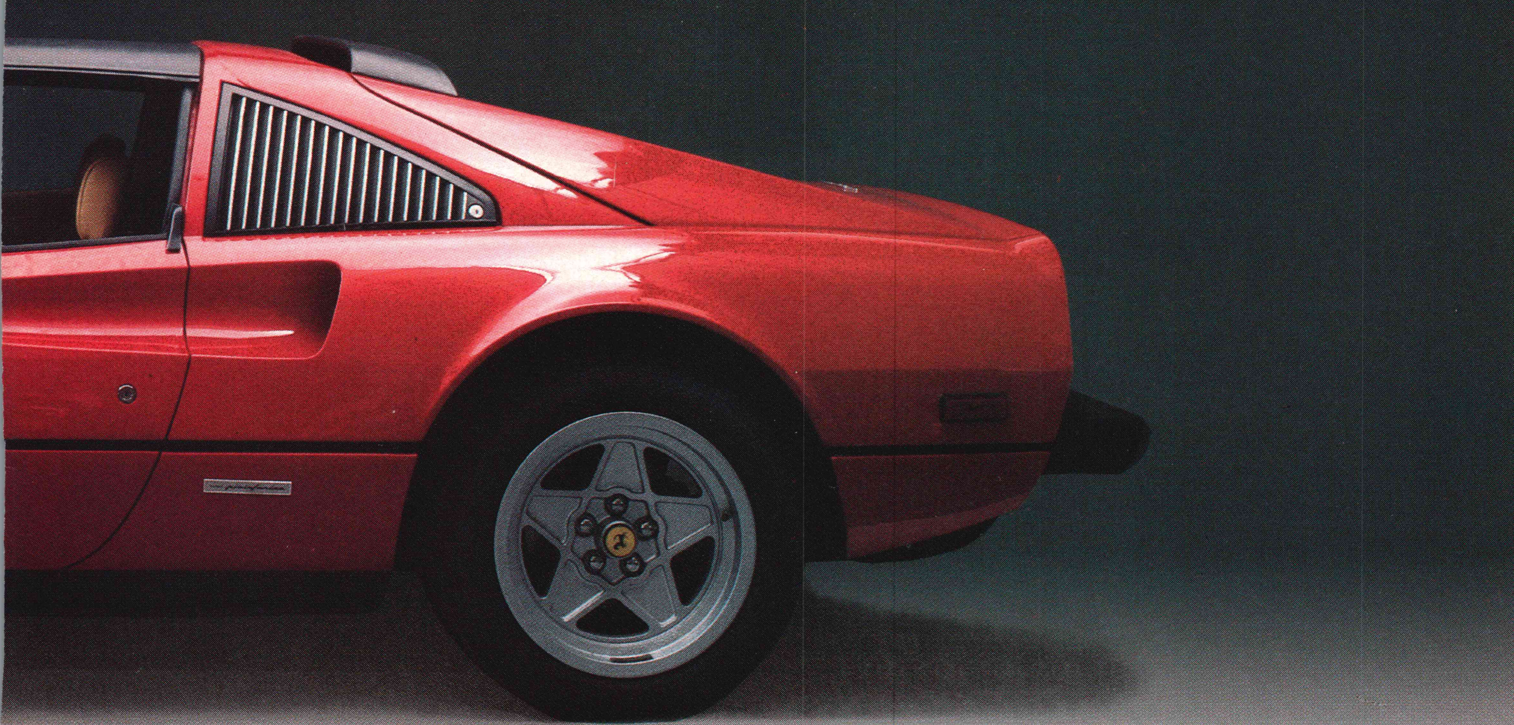


Still Fixing Bugs The Hard Way?

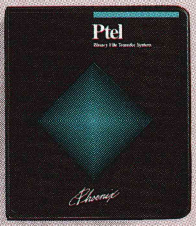
Pfix™86 Plus, the most advanced symbolic debugger on the market, eliminates the endless error searches through piles of listings. Locate instructions and data by symbolic name, using symbolic addresses. Handle larger, overlayed programs with ease.

An adjustable multiple-window display shows source and object code and data, breakpoint settings, current machine register and stack contents simultaneously. An in-line assembler allows program corrections directly in assembly language. Powerful breakpoint features run a program full speed until a loop has been performed n times.

With a single keystroke you can trace an instruction and the action will be immediately reflected in source, object, data, stack, and register windows. Another key begins a special trace mode that executes call and loop instructions at full speed. Designed to work with both Plink86 and MS™ LINK linkage editors. **$395.**



ies by Phoenix.

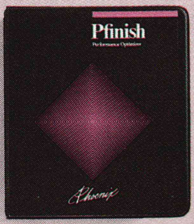


Get The Lead Out Of Binary File Transfer.

Ptel™ is the universal binary file transfer program for MS-DOS 2.0 or 3.0. You can move binary files fast and accurately. Upload or download groups of files from Bulletin Boards or remote computers. Move files between dissimilar machines and operating systems.

Ptel's advanced binary protocol, Telink, offers better-than-Modem7 accuracy and performance. Faster transfer speeds. An on-screen update of error correction, blocks, transferred, and time to complete.

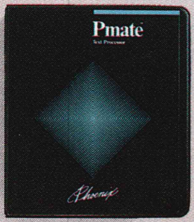
Includes popular Modem7 and XModem protocols. With Checksum or CRC. Plus Kermit and ASCII. **$195.**



Maximize Your Program's Efficiency.

Pfinish™ delivers the fastest running programs possible. This performance analyzer lets you ''zoom in'' on the inefficient parts of your program. Whether written in assembly language. C. Pascal. Fortran. Even Basic. Unlike profilers available today, Pfinish understands the structure of your program and reports the amount of activity and time spent in its subroutines or functional groups. Pfinish analyzes both overlaid and memory resident programs. Down to the instruction level. Reports are displayed. Stored on disk. Or printed out. In tabular form or histograms.

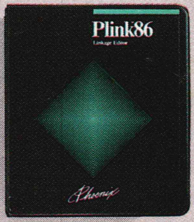
Do a dynamic program scan. Identify the most frequently executed subroutines. Find inefficient code that costs your program valuable time. Rank subroutines by execution frequency. **$395.**



Why Work With A Primitive Editor?

More than a powerful editor, Pmate™ is a text processing language. An emulator of other editors. A language-specific editor for C, Pascal, and Fortran. Pmate™ can even run in the background!

You get full-screen, single-key editing. Ten editing buffers. Horizontal and vertical scrolling. A ''garbage stack'' buffer. A built-in macro language with variables, control statements, radix conversion, tracing and 120 commands that you can group and execute with a single keystroke. **$225.**



Why Squeeze Your Program More Than You Have To?

The Plink™86 overlay linkage editor brings modular programming to 8086/88-based micros. Write large and complex programs without worrying about memory constraints. Work on modules individually, link them into executable files. Use the same module in different programs. Changes the overlay structure of an existing program without recompiling. Use one overlay to access code and data in other overlays.

Plink86 links Intel-format modules. **$395.**

Call (1) 800-344-7200. In Massachusetts (617) 762-5030. Or, write.

Phoenix

Phoenix Computer Products Corp.
1420 Providence Highway Suite 115
Norwood, MA 02062

Circle **no. 122** on reader service card.

by Mike Swaine and
Bob Albrecht

Liberating the Mac

The Apple Macintosh embodies a frustration and an irony: an irony for anyone who knows the history of the company and observes that it is IBM rather than Apple that is selling an open-architecture machine; a frustration for anyone who knows the sense of power and satisfaction that comes from popping the cover off an Apple II to plug in a card and has come up against the closed design of the Macintosh.

After setting an example for open design with the Apple II, Apple has created in the Mac an appliance computer "for the rest of us," but some of the brightest people feel particularly excluded from that "us." That's unfortunate, since the Mac implements a brilliant user interface using a powerful and well-designed microprocessor. But for experienced programmers, that interface can get in the way, and the bulk of the bandwidth of the microprocessor is swallowed up by the graphics. What Apple has done in producing the Mac is remarkable, but one of the things it has done is to make evident all the possible things it didn't do. Some programmers seem to feel that the Mac has too much potential to be left to Apple, and have put forth various schemes for liberating the essence of the machine for their own version of "us." This month's Realizable Fantasies are all schemes for the liberation of the Mac.

The Hacker's Mac

Lee Felsenstein has been talking for the past six months about something called "the hacker's Mac." The hacker's Mac is not in essence a commercial product or even a proposed commercial product, but a project. Lee's intent appears to be to engage the imaginations of engineers and programmers who may grudgingly admit that they like the Mac, but who know in their hearts that they would have done it better.

The hacker's Mac is a consensus design project with the objective of developing a computer that does all that the Macintosh does and more. Anyone who cares to contribute ideas to the design, may. Clearly that's not a way to develop a commercial product; for one thing, nobody could hope to compete commercially head-on with Apple without Apple's resources and Apple's Macfactory. Well, sure. The endeavor may be, though, a way to learn the design of the Mac so intimately that you can find ways to extend, improve upon the technology. All the Mac experts a year from now will not be Apple employees and alumni. Did Burrell Smith and Andy Hertzfeld and Bill Atkinson get something more than money out of their work on the Mac? Sure, and Lee seems to think that there's no reason that others can't get an education from creating a Maclike computer on their own—or with some help from friends. So he's kicked off the hacker's Mac project.

As Lee has pointed out to me, a monthly magazine is an inappropriate medium for issuing progress reports on an ongoing project, particularly one that could move along as quickly as this one could. I shall report, then, only that Lee introduced the hacker's Mac project at the Hacker's Conference in the Marin headlands last fall and led the *Homebrew Computer Club* in sketching out the technical desiderata of the machine in January. It is February as I write this; suffice it to add that you can follow the progress of the hacker's Mac project on line by calling Gordon French's BBS (the French Connection) at 408-736-6181. By press time, this board should have hacker's Mac info; you can also learn more about the project by sending a self-addressed stamped envelope to Lee at Golemics, 2600 10th, Berkeley CA 94710.

Free the ROM 64

Steve Jasik, a veteran of Control Data and Sorcim, has written a program he calls MacNosy, "the disassembler for the rest of us."

MacNosy (Nosy for short) is an interactive disassembler for 1M Lisa systems and Fat Macs. According to Steve, it lets programmers recover the source code, or a reasonable facsimile thereof, for any Macintosh application program, ROM or "code" type resources on the "System" file. It splits programs into procedures and referenced data blocks, and maintains journal files of its activity for later playback or trading with your friends. Steve brought a copy around to *DDJ* and demonstrated its capabilities, and, whatever Nosy's strengths and weaknesses, it was clear that Steve, like the hacker's Mac consensus designers, is working to liberate the Mac.

Here's how Steve views his project: "Nosy is more of a decompiler than a disassembler. Its purpose is the recovery of source code in a human-readable format that reflects the intent of the code's author. To achieve this goal, it treats the program it is disassembling as a tree of procedures. It does a tree walk to locate the regions that are code; the remaining regions are marked as data. At present it does not gather enough information to determine which data areas are really code (like procedures passed as parameters), and human help is then needed to give this information to it. One then repeats the tree search and continues, until the

program is in its original format, or some reasonable representation."

Steve sees the disassembler as appropriate for anyone who wants to understand 68000 and Macintosh programming techniques by studying others' code and for developers who want to "find out what the ROM routines are doing to or for them." He also sees it as a tool for people who want to remove copy protection code from programs they want to run from a hard disk, and for programmers to find out how easy it would be for someone to remove copy protection code from their programs.

You can get the current version of Nosy for $70 or maybe less from Free the ROM 64 in Menlo Park, California.

Blue Light Special

Then there's the machine everybody's calling the Jackintosh: Jack Tramiel's Atari's ST's version of who the rest of us are. The ST, running Digital Research's Maclike GEM user interface, will be sold in K-Marts at a price that will take desktop icons out of Yuppiedom for the first time. Ron Jeffries (of *The Jeffries Report*) is impressed with GEM and with the ST, and seems to think that the machine will do well. Will it? Will icons play in Hamtramck? Only The Marketplace knows for sure. (And is it true that DRI is convinced that MSDOS machines represent the real market for GEM? Sigh.)

"Liberation" is a slippery word; it has taken us from considerations of open *vs.* closed architectures to K-Mart Blue Light Specials in the Computer Department and Jack "business is war" Tramiel leading a war of interface liberation. Anyway, although Atari's marketing strategies are a different kind of liberation of the Macintosh from reverse-engineering the machine or disassembling the ROMs, the ST is evidence that the user interface that Apple liberated from the workstations of Xerox PARC isn't waiting for the Mac to carry it to the heartland.

Virtual Slots

But is all this Apple-tweaking really necessary? As Tiny BASIC author Tom Pittman said in these pages two months ago, the Mac presents us with a new environment for hacking; if a few years ago you could accept the "closed architecture" of integrated circuits as opposed to discrete components, why not accept today the closed case of the Mac and concentrate on exploring its rich software environment? You can't plug boards into the Mac, but you can add desk accessories. And external enhancements.

And therein (in externals) lies the basis for Apple management's recent claim that Apple has opened the Mac. Apple's new LAN, goes the argument, gives the Mac "virtual slots."

The network is certainly open. Apple has taken the lid off AppleTalk for developers, and many have taken advantage of that fact. 3Com, for example, has developed an AppleTalk-to-Ethernet link. The protocols for AppleTalk are all public. Apple is making the multilevel network open at all levels, in the hope to "spur developers to provide innovative products for AppleTalk." It seems to be happening.

Do virtual slots offset what some see as the Mac's deficiencies as a developer's dream machine? Is Apple the true liberator of the Macintosh? Or is that just Apple's fantasy?

Free-floating Fantasy

And now for something completely different. Everybody wants to define or redefine "hacking." Steven Levy devoted a book to getting a handle on the term; the subsequent Hacker's Conference wrestled with the definition; old-time programmers wonder whether the term has been irrenewably tarred by the brush of illegal-entry pranks and mischief. Perhaps we need to liberate the word from definition. Undefine it. Perhaps this liberating will happen as a matter of course, whatever we do. Perhaps liberating the word "hacking" is some part of what "hacking" (or "to hack") means.

Consider the denumerable set of all possible meanings of the term "to hack." Call the cardinality of this set n. Proposition: n is the cardinality of the natural numbers, i.e., infinity (in the aleph-one sense). Defend the proposition by enumeration.

Well, let's see. (1) to hack is to undefine "to hack." (2) to hack is to do more with less. (3) um, reader involvement here is welcome.

"Hackers don't grow older; they just get lazy or preoccupied or enmeshed in the bottom line. What was done once can be done again." — *Laran Stardrake.*

Micro Subscribers Please Note:

MacNosy is not the first disassembler for an Apple machine we've published. In 1976 we printed a description and a listing of a 6502 disassembler, both written by Steve Wozniak and Allen Baum. People to whom good ideas come easily tend to be free with their ideas, and Woz, who recently left Apple over philosophical differences, has always been a believer in sharing knowledge. Over the years he has given away several pieces of 6502 code through the pages of *Dr. Dobb's*, including the disassembler, floating-point arithmetic and conversion routines, a (very simple) number-guessing game, and BASIC routines for renumbering and appending.

It's unlikely that we'd reprint the articles in *Dr. Dobb's*. We want to run *current* 6502 material, and these pieces may be most appealing for their historical interest (although it's instructive to read Woz's code). But we could put together a reprint package if there's enough interest. We'd have to charge enough to cover the postage and printing and handling—probably three dollars. But we could throw in Andy Hertzfeld's *Lazarus*, a program for resurrecting BASIC programs on the Apple II.

If you'd like to have such a reprint, write to The Woz Papers, c/o this magazine, and let us know. If interest is great enough, we'll do it.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 190.**

DR. DOBB'S CLINIC

by D. E. Cortesi

DRI Drops Homebrew Systems

An item in a recent *BUSS* gave us a mild panic. It seemed to say that Digital Research had withdrawn CP/M Plus from sale. We called some people we know in Pacific Grove and found that the news wasn't that bad. CP/M Plus is alive; it's homebrewing that's been given up for dead.

DRI used to sell its various DOSses in generic form: the base code, unconfigured to any hardware, needing a complete BIOS to run. Lots of hobbyists bought CP/M 2.2 that way; we and a few others bought CP/M Plus in the same form.

No more. Today, most of DRI's retail sales are of preconfigured systems (for the IBM PC, mostly) and of languages such as DR LOGO, which need no configuration. The generic packages aren't moving, our source said; they burden the inventory and bring in no revenue. So, DRI is phasing out retail sale of all generic systems. An added-value vendor who wants to package and resell a DOS in quantity may license it, of course.

But sales of unconfigured systems to individuals are ending. Wait, we protested, there are still a few of us who want to roll our own. What about us? Well, DRI doesn't mind you getting a copy of a generic system, but they're determined to shed the inventory burden. They are "talking to" third parties who may want to take over the sale of generic systems as a sideline, but nothing has been settled.

If you think you might ever want to buy a single copy of CP/M Plus or CP/M 68K or Concurrent CP/M in generic form, write to the marketing folks at Digital Research (P. O. Box DRI, Monterey, CA 93942) and tell them. They think all the homebrewers have bought PCs and retired.

Criticism of Modula-2

One of the oldest and most vigorous SIGs sponsored by the ACM is SIGPLAN, the Special Interest Group on Programming Languages. Its monthly newsletter, *SIGPLAN Notices*, has carried some of the most influential language papers ever published; yet because it isn't a formal journal, it also carries some of the funniest and some of the most controversial papers as well.

One item in the December 1984 issue falls into the latter category. "Some Concerns About Modula-2" by David V. Moffat of North Carolina State University appears to be intended as an antidote to some of the gushier praise lavished on Modula-2 of late. Some of you may not have access to *SIGPLAN Notices*. Because we wouldn't want you to miss out on a solid linguistic controversy, we thought we'd summarize Moffat's points here.

His first point is that, as a result of its relegating all I/O to procedures written in the language itself, Modula-2's I/O syntax is very low level compared to other languages. It "forces the programmer to use a different procedure for each type, for each number of parameters, and for each default," Moffat writes, whereas languages that contain built-in I/O statements, however inconsistent (think of Pascal's pseudo-procedure WRITE), are a lot easier to use.

He says that the standard I/O modules are, in his opinion, "a hodgepodge of oversights, inconsistencies, duplicate names, and hasty patches." And he gives examples: you cannot write real numbers to the terminal while accessing a file with the standard module InOut; the I/O modules export many similar names, "so the simple Pascal . . . statement 'write(s)' becomes 'Terminal.WriteString(s).' "

Moffat's third point is an involved analysis of the portability of modules as object code. Because of the linker's version checking, and because any implementor will have recompiled the standard I/O modules one or more times, he concludes that the modules will not port, which forestalls substantial traffic in precompiled modules. But he doubts that distribution of modules in source form will work either, partly because developers are reluctant to release source but also "because the 'standard' modules are not really standard. There is no guarantee, and indeed no real expectation, that a procedure in one version of a module will have the same syntax or semantics—or even exist—in another version of that 'same' module."

Moffat notes the absence of built-in provisions for varying strings, for sets large enough to cope with the alphabet, and for high-precision numeric types. And although he admits that "all of these types can be simulated in modules, an activity that is too often beside the point of the program to be written, although a hacker's delight," their absence as efficient built-in types and the presence of "lots of features for systems programming" make it "plain that Modula is a systems programming language, not a general purpose language."

Because the "effective size of the language is very great" (Moffat notes 40 reserved words, 26 standard identifiers, 130 identifiers in the standard modules, and "some implementations provide yet 200 more!"), he deems Modula-2 confusing and ineffective as a teaching language. Because it's neither general purpose nor good for teaching, he concludes that "Modula-2 is not the successor to Pascal

or a substitute for Pascal," as many have called it. He points to UCSD Pascal as "a genuine extension and improvement of Pascal."

We'd enjoy seeing your informed comments, pro or con, on Modula-2 and will print any that seem enlightening. However, if you feel moved to debate with Moffat specifically, you should read the original paper and direct your comments first to *SIGPLAN Notices*; we're just some interested bystanders.

Headless No More

A few readers wrote to commiserate with us regarding the decapitation of our Shugart 860, dramatically described in the December 1984 issue. To our surprise, they didn't think much of the Tandon half-height either. Rex Backus of St. Helena, CA, made a typical remark: "I really wonder if you were just lucky to get a working 848-2 with only one try or if Tandon is doing a better job with those drives than they did two years ago. . . . I sent two drives back for replacement before I got a pair to work." The answer is probably yes, Tandon has improved; their latest drives, we hear, are quite different from their first efforts.

However, these comments made us wonder. How many other people are suffering from such hardware problems? More to the point, why are they keeping it a secret? That's what this column is for; we *love* to collect anecdotal reports of the reliability (or lack thereof) of hardware and software. How are the Qume and Mitsubishi drives?

One reader offered us more than sympathy. Chuck Schuetz called to say that when he'd worked for Shugart he'd had a hand in the design of the 860; that he knew exactly what we were talking about; and that he'd like to fix our busted drive. He took the headless drive for a weekend and returned it working perfectly.

Chuck told us a lot about the 860. He maintains that it ended up as a really excellent drive in its final versions but allows that it took a number of MLCs (machine-level changes) to reach that state. For example, the earlier versions included a small boss to force an intentional bow in the diskette jacket; this boss had to be ground flat in later versions. In fact, most of the drive's problems related to the positioning of the diskette jacket in the frame. The drag and varying speed that were our original problems most likely were due to a maladjustment that let the eject slider bear against the corner of the jacket when the drive was closed.

Schuetz is willing to advise other people with distressed 860 drives. He can get spare parts at reasonable cost and can handle a limited number of repairs as his spare time permits. He'd prefer your initial contact with him to be by mail; write him at 3675 Desoto Avenue, Santa Clara, CA 95051.

Shadowy Printing

OK, the Intern needs more help. Our Diablo 1650, sturdy companion through zillions of words, is getting

flaky. The figure below shows the problem. It shows lots of shadow-printed Os. As you'll see if the reproduction is good, the amount of shadow varies from letter to letter. Some overstrikes are offset a little, some a lot.

The problem seems to be a slight random variation in the carrier's horizontal position. It shows the worst in shadow printing, but it can also be seen as a slightly "dirty" look in proportionally spaced text. The problem is not in radial control of the typewheel, because it can occur when the typewheel isn't turning (as in the figure), and it isn't a typewheel problem, because it affects all typewheels including new ones. It's a problem of rapid, incremental, horizontal movement; it probably has an inertial component, since running the printer at half speed (300 baud not 1200) improves the output.

We took it to a local Xerox America service franchise. They replaced the carriage and the horizontal drive motor. They claim they swapped out all the boards. At our urging, they checked the power supply under load. They kept the printer two weeks. They presented a large bill and said, "There's nothing wrong with it now."

Baloney. Shadow printing still looks awful. Are there any devilishly good printer techs out there who could tell us why?

Well-Behaved Comments

Back in October we outlined a method by which a C compiler could (1) detect which functions absolutely couldn't be recursive and (2) allocate their local variables in common pools of static storage. Several readers were moved to comment on the scheme.

David S. Tilton, whose letter last summer started the whole train of thought, found a hole in our scheme. Although we set out to identify all the well-behaved functions (WBFs), we ended up identifying only one of them. Tilton says, "Consider three functions which each make one call. A calls B, B calls C, and C calls B. Clearly B and C are part of a recursive loop and are therefore not WBFs. It is also clear that B and C constitute a closed set; the call from A to B cannot be part of a recursive loop, and A is definitely not recursive." But our method would not classify it as a WBF.

In part, that's because we merged two distinct kinds of ill-behavior. We viewed as "potentially recursive" functions that called themselves or each other (functions B and C in Tilton's example) and also functions that called external or pointer-based names. A compiler in principle could recognize the limited scope of the former (although neither we nor Tilton can think of a simple algorithm to do so), but it's a peculiarity of C that the latter behavior is intractable. If A calls B, B calls C, and C is declared external, then we must assume that the external function C *might* call either A or B or both—we just can't tell.

James Jones of Norman, OK, pointed out that the method might be simpler if we reversed it, searching for the ill-behaved functions (IBFs) instead of the well-behaved ones. A direct IBF (DIBF) would be a function that was itself objectionable—declared external, called itself, or called a pointer. Then a general IBF would be any function that called a DIBF at any level. In pseudo Prolog:

```
IBF(f) if member(f,DIBF).
IBF(f) if CALLS*(f,g) and
    IBF(g).
```

Here CALLS* signifies the transitive closure over the CALLS(a,b) relation. We can find the transitive closure in order n-squared time, Jones reminds us, using Warshall's algorithm. (The final phase of the method we described bore a strong resemblance to Warshall's algorithm.) Jones cites David Gries' *Compiler Construction for Digital Computers* (Wiley, 1972, page 39) as a source for Warshall's algorithm, but heck, it's easy to describe.

Suppose i and j are both integers between one and some maximum n, and suppose that you've defined a relationship R(i,j) that is true for some pairs (i,j) and not others. In our case, the relationship is CALLS(i,j), and it's true when function i calls function j. We can express the complete relationship in the form of a matrix of bits, B, such that B[i,j] is zero where R(i,j) is false and one where it's true.

We really want not CALLS(i,j) but CALLS*(i,j), the transitive closure of CALLS: the relation that is true if function i calls function j directly *or* through any number of intermediate functions. Warshall's algorithm to convert matrix B into matrix B* (using pseudo-C notation) is:

```
for(i = 1; i <= n; i++)
    for(j = 1; j <= n; j++)
        if B[j,i] then
            B[j,*] |= B[i,*];
```

The innermost line signifies the bitwise ORing of row i into row j; it could be spelt out as:

```
for(k = 1; k <=n; k++)
    B[j,k] |= B[i,k];
```

However, the algorithm is n-squared only when the rows of B can be ORed as single objects, preferably with a fast assembly subroutine.

Jones and Jim Howell of San Jose,

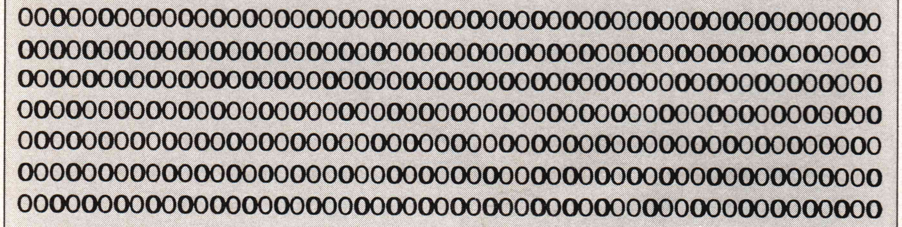


Figure
The irregular shadowing produced by the Intern's printer.

CA, both commented that the Motorola 6809 CPU shouldn't be tarred with the same brush we used for the 8080; the 6809 handles stack-allocated variables very well and wouldn't benefit much from their being static. Howell notes that, lacking a smart compiler, C programmers can create their own data type

```
#define LOCAL static
```

and use it in the functions that they know to be nonrecursive:

```
fun(arg)
{
    LOCAL int i,j;
```

Then, Howell says, "when porting to a machine that can efficiently access variables on the stack, just change one line to

```
#define LOCAL auto
```

or even to

```
#define LOCAL /* nothing */
```

All the local variables are now automatic."

Both Howell and Brian McKeon of Saugus, MA, noted that our method doesn't take account of functions that could be called as the result of interrupts. Paul Canniff of King of Prussia, PA, commented that stack-allocated locals, especially arrays, conserve storage, and a compiler that makes locals static may produce programs that are too big. Clearly, any compiler that used our method would have to provide a switch to turn it off. Alternatively, the compiler could take an explicit declaration of "auto" as barring static allocation.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 191**.

MONTY PLAYS

Talk of "computer literacy" can sometimes be a mask for a botched and inhuman design.

"Computers evolved from scientific superbrains to business machines to household appliances; had they started out as video games, they would certainly have been designed with the user interface in mind."

The quotation above, from D. Verne Morland's article, "Human Factors Guidelines for Terminal Design" (*Communications of the Association for Computing Machinery*, 26(7): 484-494, July 1983) is the starting point for our case study of human factors engineering and its application to the world of microcomputer software design. We will approach this investigation by examining a popular computer game and seeing how human engineering concepts have been applied to make it simple, enjoyable, and useful. We will also look at some microcomputer software products and consider how and where we might apply these same concepts to them.

Human factors engineering is that branch of systems analysis and design that deals with the interface between the computer user and the software/hardware combination. On the hardware side, human factors engineering helps designers determine where they should place special function keys on the keyboard, what color monitor is least fatiguing, and so on. On the software side, it helps programmers to design program menus, or choices to accomplish certain tasks. Finally, human factors engineering addresses the user—problems of operator error (and how to handle these errors) and whether a computer or a program is sufficiently user friendly.

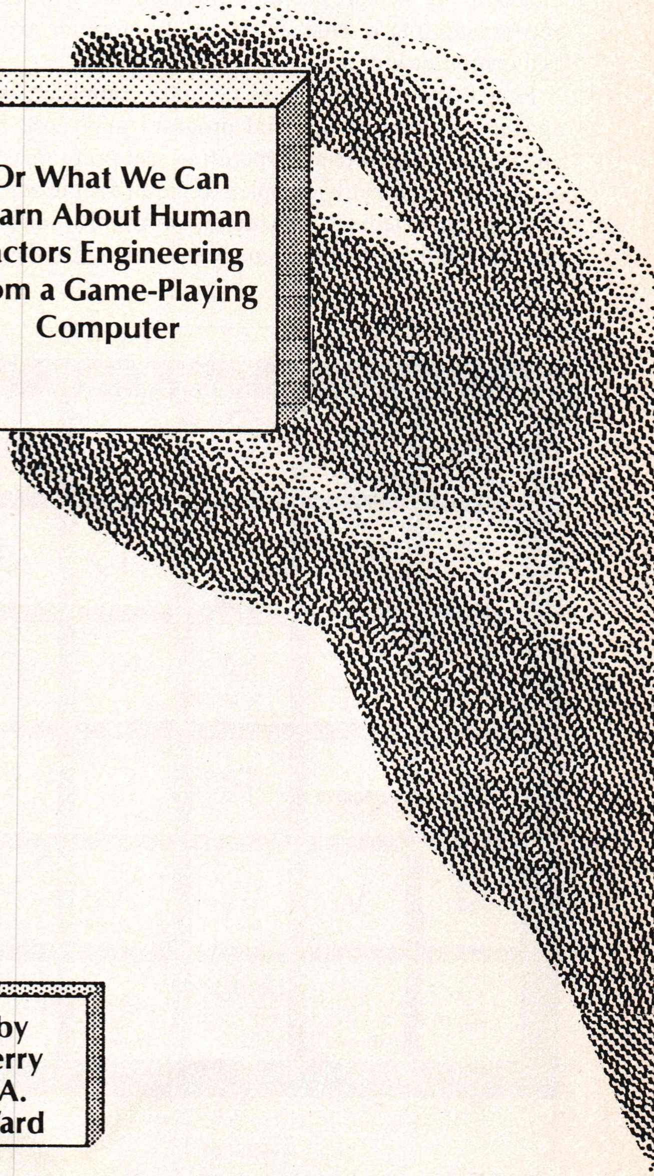
Although human factors engineering probably should be an integral part of any computer hardware or software design, the numerous variations of keyboard layout and the contorted sequences of keystrokes required to operate some programs would indicate that human engineering has *not* been applied uniformly. For every powerful, accessible, easy to use system available, many others perform as if no one ever tried using them before making them available for sale.

Scrabble is a trademark of the Selchow & Righter Company.

Terry A. Ward, Academic Computing Services, University of Northern Iowa, Cedar Falls, IA 50614.

SCRABBLE™:

Or What We Can Learn About Human Factors Engineering From a Game-Playing Computer



by Terry A. Ward

The Product

Let's look at a popular microcomputer-based game system for playing Scrabble. For those unfamiliar with its rules, Scrabble is a crossword-puzzle game that can be played by two to four people. The players form interlocking words on the Scrabble playing board using letter tiles with various point values. The more common letters such as I or E are very low scoring (1 point each), while letters such as Q are worth more (in this case, 10 points). Players compete by using their letters and the various bonus squares on the board to maximize their score while minimizing their opponents' opportunities.

The computer variation of Scrabble involves MONTY, a self-contained, game-playing computer manufactured by the Ritam Corporation. Microcomputer versions of the game are available on diskette for both the Apple and the Radio Shack computer systems and have a similar operation. Further information on the game appears at the end of this article.

Basically, the self-contained MONTY (as we'll refer to the game from now on) looks like an oversized calculator with a membrane keyboard, which has some specialized characters, and a small 4 × 8 LCD display window. Figure 1 (page 20) shows a stylized view of the MONTY keyboard and display area. MONTY measures 6 × 10 inches and weighs about two pounds. Within this small frame is packed a formidable Scrabble challenger—and an excellent example of human engineering.

The Human Elements

Human factors engineering has certain principles or axioms that are applicable to any computer application, whether it be an extensive payroll program or a game-playing computer. The first rule or axiom is to identify your system users. This may seem obvious when stated so baldly, but many software designers neglect to do this. Consider any number of word processing or text editing software systems. The hardware includes a keyboard and a video display screen, and a great many software products for text editing refer to "screens" of information. However, screens are a convenience only to the hardware or to the programmer. Any user of word processing does not deal with screens. Writers write *pages* of books, secretaries write *pages* of letters, and even programmers write

pages of documentation. The system is oriented toward programmer or hardware conventions that have no meaning to the word processing audience.

Designers also fail to take into account the variety of people likely to use a computer system. A program written for other programmers (such as Unix or CP/M) can be terse, concise, and not particularly user friendly. But the average computer user who desires simply word processing capabilities or spreadsheet functions should not have to deal with obscure names for common functions. Programs for these users must provide easier access. CP/M shells (such as Unica or MicroShell) and menu-driven systems (such as the Morrow or Osborne systems) are attempts to match established programs with a new user audience. If the program is intended for everyone from programmers to the company vice president, it must provide some means of accommodating a wide spectrum of computer experience and computer literacy.

Furthermore, software should make complex tasks manageable. Obviously, a trivial program may need little, if any, human engineering. Operating real programs, however, is often sufficiently complex to require some assessment of the human element. In general, any application program can be subdivided into manageable tasks for the human operator. For example, WordStar, an intrinsically complex program, has a mind-boggling variety of commands, but remains a top-selling word processing program. This is because WordStar makes information available in the form of help screens to anyone who fails to remember the arbitrary keystrokes required for operation; sophisticated users may omit the help screens. This on-line help alleviates the problem of program complexity. As we will see in our case study of MONTY, this game uses the keyboard itself as a reminder for its functions.

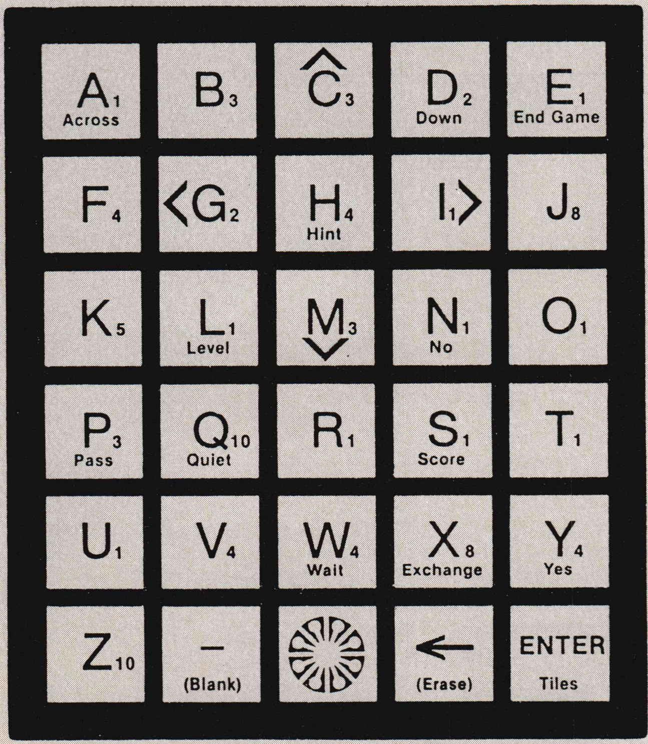


Figure 1
MONTY keyboard layout: A 4 x 8 character LCD screen provides a window to the entire Scrabble board. The commands used in playing the game are printed in contrasting colors to make them more apparent. The point value of the letters is also reproduced on the keyboard.

Finally, programs should weigh the consequences of operator error. If the consequences are truly catastrophic (such as file deletion or program abort), the program should not take such action lightly. Release 3.0 of CP/M (also known as CP/M Plus) has instituted automatic relogging of diskettes to compensate for the common user mistake of switching disks without doing a control-C. When you do that, the dreaded "BDOS ERROR" appears and the system must be restarted or rebooted. In general, programs should try to prevent accidental disasters due to operator error. MONTY does this without appreciably slowing down the game-playing process.

Software Design Principles

We have noted that our first general axiom is to consider the human elements in the human-machine interface. The second principle of human engineering is to design the product as much as possible to meet human needs—for feedback, consistency, memory aids, simplicity, and so on. We will now consider specific rules to follow when designing software.

The first rule is to provide some means of feedback as to the program's current status. Whenever the computer is doing something that may take a few moments, the user should be aware that the machine is still working. For example, a word-processing program will often appear totally "dead" when it is doing a global search-and-replace or advancing to a distant portion of the text. This can be both annoying and disconcerting; novice users may wonder as time passes if something is wrong. A simple display of a flashing asterisk could reassure them that all is well.

The second rule is to be consistent in terminology and program design. If we require the user to enter information using the Return key, we should refer to it in our instructions as the Return key—not the Carriage Return or the Enter key. While you and I may know that carriage return, enter, and return are all synonymous, individuals new to computing will hesitate before assuming that the three names refer to the same key, as they rightly should.

A third rule is to minimize the memory demands placed on human operators. We should not expect people to use our programs if they require more memorization than they are worth. The spreadsheet SuperCalc makes help available throughout the program for users who forget the exact requirements to specify a format, for example. This frees them to handle the task of problem definition.

The fourth software design rule is to keep the program simple. Programmers are notorious for preferring the lat-

FOR THE FIRST TIME IN THE HISTORY OF THE UNIVERSE YOU CAN DEVELOP AN INTEGRATED APPLICATION THAT REALLY SINGS.

HERE'S HOW:
FRAMEWORK SOFTWARE

Framework™ is the only integrated software that contains a programming language. This means that you can use the language to create special applications which use all the features of Framework.

For the first time, you'll find it easy to design custom programs which let users outline, write, work with data and create graphs for their own special requirements, and use all Framework functions with a single set of easily-learned commands.

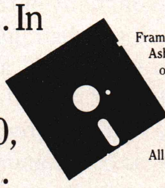
Let's say your customer is using a sales analysis program you've written using Framework. He loves the ability to draw graphs and use all the other standard Framework features. To his surprise, when the Sales Analysis graph reveals the Southern region is leading, his PC starts playing "Dixie."

If you use the Framework programming language, you'll discover the @ **BEEP** command, which lets you select both frequency and duration: @ **BEEP (440,300)** plays a pure "A" for 3 seconds. Not quite long enough to tune an orchestra, but it's the start of a melody.

Ashton-Tate has created a whole industry of vertical-market applications with its dBASE II® and dBASE III™. It's doing the same with Framework.

Climb aboard the bandwagon. Make your programs take on the beauty of the varied capabilities of Framework.

For a dealer near you call (800) 437-4329, ext. 2233. In Colorado (303) 799-4900, ext. 2233.



Software from

ASHTON·TATE™

We'll put you in control.

Circle **no. 4** on reader service card.

est and most complex versions with "all the bells and whistles," but users simply want to get the job done. The program Bank Street Writer, while not in the same league as WordStar, nonetheless provides usable word processing capabilities to many people. I have even seen a review of Bank Street Writer written by a six-year-old child—an obvious endorsement of keeping it simple!

A fifth rule is to match the program to the operator's skill level. Many software programs assume that all users are equal: everyone must venture through countless help screens or menus to accomplish even the simplest task. For example, the SCSS conversational statistics package is very user friendly, but you must answer an abundance of questions to obtain even the simplest results. To plot two variables requires close to two dozen instructions, while a similar task in the statistical package MINITAB requires but three commands. Providing help screens on request only will accommodate both novice and advanced statistical users.

Finally, our last rule for software design might be paraphrased as: "The user is always right." Programs should filter and correct user mistakes. Often user error simply indicates poor program design. For example, our word processor requires that users remember that the Home button *really* means Down Arrow. Anyone who uses the key incorrectly is obviously guilty of user error, but the program actually invites just such an error. Always maintain a user or operator orientation. We are writing programs for people to use, not purely academic exercises.

MONTY—A Case Study

As we noted above, MONTY is a hand-held computing device that can play Scrabble with up to three other opponents. Let's look at the design of this computer game with an eye toward the first axiom of human factors engineering: identify your audience.

Obviously, MONTY is designed for people who wish to play Scrabble. Beyond that, the audience may range from computer analysts to ten-year-old children to adults who are totally unfamiliar with computers. I know individuals in each of those categories who have successfully played MONTY with only minimal instruction. How is MONTY designed to appeal to this wide audience?

To begin, the keyboard contains virtually all the commands needed to play the game. MONTY commands are superimposed on many of the alphabetic keys, printed in a contrasting color to distinguish them from the alpha characters. These commands are fully spelled out: one may "exchange" tiles or place them "across" the board. This is in contrast to many microcomputer programs that would abbreviate these commands as "xchng" or "a." This user-friendly keyboard allows virtually anyone to begin playing with little if any training. MONTY has taken into account the breadth of possible users and the diversity of their backgrounds without confusing one or patronizing the other.

In MONTY, the major catastrophic errors are ending the game prematurely, exchanging tiles (and thus losing a turn), and impossible plays. When a player proposes to end the game or exchange tiles, MONTY asks the user again if he or she is sure of this request. Only upon being told "yes" does the program implement this action. When players attempt an impossible play (such as placing tiles where they cannot legally go), MONTY returns them to the beginning of their turn. If you attempt to use a letter that you do not have, MONTY informs you of this and tells you which letter is missing. In all cases, the user is placed back into a logical position from which to try again. This is in contrast to many computer programs that will abort and place you in an unknown environment (such as CP/M) upon operator error.

Obviously MONTY has followed the first axiom of human factors engineering. We will now investigate how well our game-playing computer has utilized the six rules encompassed by the second axiom: design the product to meet human needs.

Provide Feedback

MONTY can take up to three minutes to find a word and make its play. A counter that is constantly displayed on the screen provides feedback while the program is calculating. When the counter reaches zero, MONTY plays its word. This feedback assures the opponent that all is proceeding normally. Likewise, requests for information are always accompanied by a listing of expected responses (such as yes/no or across/down). This feedback ensures that the user is aware of what is going on with the program.

Be Consistent

MONTY always requests information in a similar format or style when prompting for response. An inappropriate response is always greeted with a beep, and the request is issued again. It also uses the natural terminology of its users rather than the contrived terminology of computerdom.

Minimize Memory Demands

MONTY excels in this particular design goal. Embossing the keyboard with the commands reduces human memory demands to the bare minimum. By replacing human memory requirements with good software design, the programmer frees the user to concentrate on the task at hand—beating a machine at Scrabble.

Keep It Simple

MONTY is totally self-contained. It requires that the player know only a few commands in addition to the rules of Scrabble. This is in contrast to much of the currently available microcomputer software. Programmers have a tendency to fill any available storage space with functions, relevant or not. The range of options is simply too great, and most people use only a limited subset of a machine's capabilities. Limiting the options to those most commonly used will encourage people to use the software rather than intimidate them with an instructional manual the size of a telephone directory.

Match the User's Skill Level

Here is an area where much of the available microcomput-

Level A:	frog	
	suet	
Level B:	azo	
	eta	
Level C:	viol	
	averters	
Level D:	qintar	(unit of currency)
	hegira	(a journey)
	wadi	(a normally dry stream bed)

Figure 2

MONTY offers four levels of play. Levels may be increased or decreased without restarting the game or program. Listed are typical words encountered at each level of play.

er and mainframe software succeeds. MONTY presents all players with the same information and keyboard. It then accommodates itself to player proficiency by providing four levels of play (see Figure 2, above) and adjusting the rewards accordingly. MONTY rewards particularly good plays with a brief musical selection (such as the "1812 Overture" or "Hail to the Chief"); at level D, 50 points per play are required for "Hail to the Chief," while at lower levels of play it takes a less impressive score. This feature relates to the sixth and final design principle.

Sustain Operator Orientation

This last principle is by no means the least. In fact, it may be the most important of the half dozen presented. Programs are designed for people, not other programmers. Unix, for example, is an incredibly powerful software operating system. It is, however, generally quite unsuited for the non-programmer. CP/M is likewise an excellent operating system, but with its PIPs and CON:s and LST:s it is designed for the already computer literate. This is perhaps one of the great appeals of BASIC as a computer language or environment. Everything that needs to be done can be done in the one environment of the BASIC interpreter. You write your programs, you test them, you edit them and you even save and retrieve them from one, consistent environment. This is in contrast to the typical mainframe situation where you must always know exactly where you are. Editing commands are not recognized by the job control scanner and BASIC commands are likely to result in bizarre error messages if typed in at the wrong moment. In my experience as a user consultant, this is perhaps the major complaint of most people using computers. The machine forgets that it exists for the sake of human operators and not the other way around.

Conclusions

This venture into human factors engineering via a game-playing computer should give some insight into the reasons people complain about computers and their associated software. We are quite often to blame: many programs are *not* easy to use. Computer illiteracy often reflects programmer insensitivity to people's needs and desires. Although microcomputer sales are still quite healthy, I wonder whether we will see increasing disenchantment with software that doesn't work. MONTY and other well-designed programs can provide a starting point to enter the world of software engineering for human beings.

"MONTY Plays Scrabble"

Manufacturer: Ritam Corporation
P. O. Box 921
Fairfield, IA 52556
(515) 472-8262

Format: Apple II, II Plus, or IIe; Radio Shack TRS-80 Model 1, 3, or 4; self-contained version

Price: Apple, $39.95; Radio Shack, $34.95; self-contained, $129.95

Description: MONTY has a basic understanding of Scrabble strategy and, combined with a vocabulary of up to 44,000 words, can provide a formidable opponent to word game enthusiasts.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 192.**

Shortcut to SCISTAR: For Prowriter Users

by Amer W. Nelson

Thomas and Fletcher's excellent article, "SCISTAR: Greek and Math Symbols with WordStar" (*DDJ,* August 1984), taught me what I needed to know to incorporate those symbols, as well as super/subscripts and underlining, into my writing. The real advantage of Thomas and Fletcher's method is their use of a universal command set common to all printers. The drawbacks, as I see them, are that MailMerge must be used and that a "hunt and peck" typist like me requires a lot of time to produce files like SCI.CMD and SCI.DAT.

The shortcut I will describe has the disadvantage of being tailored strictly to the Prowriter. Nonetheless, I believe that experienced users who understand Thomas and Fletcher can make those changes necessary to incorporate the shortcut in their own computer/printer configuration.

As Table 1 (page 25) shows, I have retained or only slightly modified many of Thomas and Fletcher's WordStar patches for the Prowriter. I elected to use WordStar's own boldfacing (hence, no patch for that feature), and I chose pica (10 CPI) to be my standard print with compressed 17 CPI pitch as an alternate. I used the command ^PQ—related to USR1:—to enter the Prowriter's Incremental Print Mode before typing special characters and the command ^PR—related to USR4:—to return to the Prowriter's Logic Seek Print Mode (see the model 8510A user's manual, page 46).

Further information on how to incorporate Greek and math symbols into WordStar files.

These drawbacks caused me to search for a shortcut to SCISTAR that my wife and I could use on our personal Osborne1/Prowriter system. I am a security analyst with a small investment company, and my wife is an EEG technologist. Because neither of us produces lengthy scientific papers, our use of Greek characters, math symbols, and super/subscripts is limited. (I use an occasional β to denote the relative volatility of a stock, an occasional M_1 in reference to the money supply, and a few superscripts for footnotes; my wife makes use of the Greek characters α, β, δ, κ, μ, and θ to denote brain wave patterns and Ω, the symbol for ohms of resistance.)

Amer W. Nelson, 12030 36th Ave., NE, Seattle, WA 98125.

It's particularly important to imbed your ^PQs and ^PRs between paragraphs. When the Incremental Print Mode is on, printing is unidirectional, but when operating in the Logic Seek Print Mode, printing is bidirectional. If you try to bounce back and forth between them in the middle of a paragraph, you may find one line printing on top of another. To avoid such a problem and to conserve paper, I make it a general rule to enter the Incremental Print Mode before typing any paragraph where I intend to use special characters (be

they Greek, math, or super/subscripts) and to return to the Logic Seek Print Mode at the end of that paragraph. Because I use WordStar's boldfacing and underscoring feature—the latter through RIBBON: and RIBOFF:—I need not change print modes just to print in boldface or to underline.

Table 2 (page 26) contains specific commands that produce special characters: Greek, math, super/subscripts, and a few others. Remember, however, once you are through using one of the special characters and wish to return to English characters, you must type a ^PE. Thus, to print something like πr^2, you would type ^PW/^PEr^PWQ^PE. Similarly, to print a chemical representation of molecular composition like H_2SO_4, you would type H^PV^PWQ^PE^PVSO^PV^PWS^PE^PV.

Also remember that you cannot mix boldface and underlining with special characters on the same line. As Thomas and Fletcher point out, the Prowriter will print an alpha when it receives a 20H.

One advantage of my shortcut is that you can print the Greek gamma without jumping through hoops. Without using MailMerge, you can send a " (22H) as a variable. Moreover, if you're trying to type eight or nine pages with only a few special characters imbedded in the text, you can print out your file much more quickly in the bidirectional mode, using the unidirectional (Incremental Print) mode only as needed.

Table 3 (page 27) contains some additional information, unrelated directly to this article, that may nevertheless be of interest to Osbornel users. If you're interested in learning C language programming, it would be nice to know how to type (and print) a { or a }.

This works too! (Refer to page 29 of the August 1984 *Dr. Dobb's Journal.*) Typing x ^PT ^PW (space) ^PE ^PT (space) ^PWF ^PE (space) ^PW!Q ^PE is an inequality. produces the following print:

$x^\alpha \geq \beta^2$ is an inequality.

Nice, huh?

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 193**.

Incremental Print	USR1:	ESC[	02H,1BH,5BH	^PQ
Logic Seek Print	USR4:*	ESC]	02H,1BH,5DH	^PR
Special Characters	USR2:	ESC&	02H,1BH,26H	^PW
ASCII (English)	USR3:	ESC$	02H,1BH,24H	^PE
Underscore On (toggle)	RIBBON:	ESCX	02H,1BH,58H	^PY
Underscore Off (toggle)	RIBOFF:	ESCY	02H,1BH,59H	^PY
Superscripting &	ROLUP:	ESCr(LF)	03H,1BH,72H,0AH	Use ^PVS or ^PTS
Subscripting	ROLDOW:	ESCf(LF)	03H,1BH,66H,0AH	(toggles)
Normal <CRLF>	PSCRLF:	ESCf(CR)(LF)(LF)	05H,1BH,66H,0DH,0AH,0AH	
Initialization String	PSINIT:	(CR)ESCT 12	05H,0DH,1BH,54H,31H,32H	
Alternate Print	PALT:	ESCQ	02H,1BH,51H	17 CPI
Standard Print	PSTD:	ESCN	02H,1BH,4EH	10 CPI

*You may find my order strange, but I associated ^PR with *Return* to Logic Seek and ^PE with *English*.
(I did not use PSFINI:. I just made sure that I was in Logic Seek Mode at the end of printing.)

Table 1.
Patches installed on WordStar disc for use with the Prowriter

Type:	To get:	
^PW<SPACE>	α	(alpha)
^PW!	β	(beta)
^PW"	γ	(gamma)
^PW#	δ	(delta)
^PW$	ϵ	(epsilon)
^PW%	ζ	(zeta)
^PW&	η	(eta)
^PW'	θ	(theta)
^PW(	ϑ	(iota)
^PW)	κ	(kappa)
^PW*	λ	(lambda)
^PW+	μ	(mu)
^PW,	ν	(nu)
^PW–	ξ	(xi)
^PW.	ο	(omicron)
^PW/	π	(pi)
^PW0	ρ	(rho)
^PW1	ς	(sigma)
^PW2	τ	(tau)
^PW3	υ	(upsilon)
^PW4	ϕ	(phi)
^PW5	χ	(chi)
^PW6	ψ	(psi)
^PW7	ω	(omega)
^PW8	Δ	(upper-case delta)
^PW9	Γ	(upper-case gamma)
^PW:	Σ	(upper-case sigma)
^PW;	Λ	(upper-case lambda)
^PW<	Ω	(upper-case omega)

Most upper-case Greek characters can be printed by typing normal upper-case English characters: upper-case alpha is A, upper-case beta is B, and so on. For those upper-case Greek characters that the Prowriter can't handle, like pi and phi, just leave two spaces and do them by hand.

When you are through with Greek characters or any other special characters, type a ^PE. For instance, to print $\phi\upsilon\kappa\ \alpha\ \delta\upsilon\kappa$, type ^PW43)^PE <space>^PW(space)^PE<space>^PW#3)^^PE.

Other available symbols (characters) include the following:

Type:	To get:	
^PW>	√	(square root sign)
^PW?	□	(upper square)
^PW@	↑	(arrow up)
^PWA	↓	(arrow down)
^PWB	←	(arrow left)
^PWC	→	(arrow right)
^PWD	±	(plus or minus)
^PWE	≠	(not equal)
^PWF	≥	(greater than or equal)
^PWG	≤	(less than or equal)
^PWH	≈	(approximates)

Table 2.
Special Character Commands

(Table 2 continued)

Type:	To get:	
^PWI	·	(dot for multiplication sign)
^PWJ	⊕	(earth symbol)
^PWK	∞	(infinity)
^PWL	∴	(therefore)
^PWM	½	(one half)
^PWN	¼	(one quarter)
^PWO	°	(degree or super zero)
^PV^PWO	$_0$	(sub zero)
^PWP	^{1}	(super 1)
^PV^PWP	$_1$	(sub 1)
^PWQ	^{2}	(super 2)
^PV^PWQ	$_2$	(sub 2)
^PWR	^{3}	(super 3)
^PV^PWR	$_3$	(sub 3)
^PWS	^{4}	(super 4)
^PV^PWS	$_4$	(sub 4)
^PWT	^{5}	(super 5)
^PV^PWT	$_5$	(sub 5)
^PWU	^{6}	(super 6)
^PV^PWU	$_6$	(sub 6)
^PWV	^{7}	(super 7)
^PV^PWV	$_7$	(sub 7)
^PWW	^{8}	(super 8)
^PV^PWW	$_8$	(sub 8)
^PWX	^{9}	(super 9)
^PV^PWX	$_9$	(sub 9)
^PWY	$^($	(super open parenthesis)
^PV^PWY	$_($	(sub open parenthesis)
^PWZ	$^)$	(super close parenthesis)
^PV^PWZ	$_)$	(sub close parenthesis)
^PW[	$^+$	(super plus)
^PV^PW[	$_+$	(sub plus)
^PW\	$^-$	(super minus)
^PV^PW\	$_-$	(sub minus)
^PW]	$^.$	(super period)
^PV^PW]	$_.$	(sub period)
^PW^*	*	(super asterisk)
^PV^PW^*	$_*$	(sub asterisk)
^PW_	$^/$	(super slash)
^PV^PW_	$_/$	(sub slash)

*The trailing ^ is *not* a "control." It is a caret, the upper-case 6.

Type:		To get:	
<CONTROL> equal sign	^=	`	(a back accent or grave)
<CONTROL> comma	^,	{	(a left-hand brace)
<CONTROL> period	^.	}	(a right-hand brace)
<CONTROL> slash	^/	~	(a tilde as used in Spanish)

The left- and right-hand braces are particularly useful to those interested in learning C programming.

Table 3.
Some Characters not on the Osborne1 Keyboard

fx80char: A Character Editor for Epson FX-80 Printers

by David D. Clark

When Epson announced the FX-80/100 series of dot matrix printers, microcomputer users were attracted immediately because of the variety of desirable features offered at a low price. One of the things that attracted me was the ability to redefine the standard character font. Such a capability is useful, if only because technical papers almost always seem to require characters that your printer normally can't print.

The FX-80 also makes a good draft printer. If you prepare a document for printing with a letter quality printer such as a Diablo or NEC, printing early drafts can be tedious because the printers are so slow; the normal character set of a dot matrix printer, however, may not correspond to the characters available on the print element used with the letter quality printer. With the Epson, you can redefine the dot matrix characters to correspond exactly to those on the print element of the slow printer. In my case, I have access to a letter quality printer at work and use the FX-80 at home. By redefining the standard characters, I can get a reasonable facsimile of how the document will look when printed at my office while still working at home.

The program described in this article, fx80char, is designed to aid in the development of alternate character sets for use on the Epson FX-80/100 series of printers. Written in the Eco-C version of C for CP/M, the program provides a screen-oriented character editor, the ability to read and write disk files containing character set definitions, and commands to download new character sets to the printer.

The most underused printer feature may be the redefinable character set. This tool makes the feature more usable.

Commands

After fx80char is started, it displays a menu of commands. To select a particular command, you just type the highlighted character. It is not necessary to hit a <Ret> after typing the character. This section discusses each of the commands in alphabetical order.

The Activate command is used to toggle the use of the printer's ROM or RAM character set; it is assumed that the ROM set is active when the program starts. The first use of the Activate command causes the printer to use the character set stored in RAM for subsequent printing. Successive uses of this command will toggle between the two character sets. The program always displays the active character set after executing this command.

The Edit command is used to create or alter a particular character. When you enter the character editing routine, you will be asked for the number of the character to edit. This number is simply the decimal number associated with the ASCII character.

David D. Clark, 126 Birchview Dr., Piscataway, NJ, 08854.

A number is requested because sometimes you can't enter the desired character from the keyboard.

After you enter the number of the character, the character itself, as currently defined, is displayed on an enlarged "easel" for editing. You can move the cursor around within the easel, as well as "flip" the state of the individual elements. Elements displayed in reverse video represent the pins of the print head that will be fired and where ink will appear when the character is printed.

There are two equivalent groups of movement commands. The first is similar to that used by WordStar: Ctrl-E moves up, Ctrl-D moves right, Ctrl-X moves down, and Ctrl-S moves left. You can use Ctrl-F to flip the state of the element upon which the cursor rests. The second allows you to use a numeric keypad for editing: 8 moves the cursor up; 6 moves it right; 2 moves it down; 4 moves it left; and you can toggle the current element by tapping 5. The two groups of keys may be mixed at any time.

When alterations to a character are complete, typing a Q or Ctrl-Q updates the character definition and exits the editing function. During the update process, the character is converted from the form used for convenient display to the actual form used when sending definitions to the printer. The definition is checked for two types of errors: characters nine elements high and characters with consecutive printing of horizontally adjacent dots. Because the FX-80/100 can print only characters that are eight or fewer cells in height, definitions with nine cells are not allowed. Similarly, although eleven horizontal positions are available, the FX-80/100 can print no two consecutively. If an inadmissable definition is detected during the conversion process, a message to that effect is displayed. No further action, however, is taken: the character will be converted but the definition will not print correctly. Attempting to send such a definition to the printer will produce indeterminate results.

The Initialize command is used to clear the buffer that holds the character definitions in the program's memory. This command has absolutely no effect on the printer or on the contents of its memory; it simply inserts housekeeping information into the buffer and sets all definitions to blanks. This command gives you a "clean slate" to work with. It is a good idea to execute this command before building a new character definition file from subsets of previously defined files on disk (see the Read command below).

The Load command loads the *printer* RAM with the contents of the *printer* ROM. It does not transfer data between the computer and the printer. It is useful primarily to check the proper functioning of the printer's RAM. You also can use it in conjunction with the Overlay command to replace parts of the standard character font.

The Overlay command is used to redefine all or part of the printer's character set using definitions from the program's character buffer. This command displays a menu with options to redefine all characters, a range of characters, or a single character.

If you request that a range of characters or a single character be redefined, you must select the limits of the range or the single character as decimal numbers, just as in the Edit command. If the selected characters are not within the range of legal characters (0–225), or if the low end of a range is greater than the upper end, an error message is displayed and no characters are changed.

The Print command is used to print out the entire character set. After you have Overlayed the printer's RAM character set with newly defined characters, use this command to see what they look like. Remember, to be printed, redefined characters must be Overlayed in the printer's RAM, and the RAM character set must have been Activated.

This command exposes one of the program's foibles. Character 127, the alternate zero, is not printed: in its place is a blank. I couldn't figure out how to print the alternate zero under program control. You must set a switch inside the printer itself (see your printer's user manual).

The Read command is used to retrieve previously defined fonts from files on the disk. You first are asked for the name of the file. After you enter the name, a new menu is displayed giving you the option of reading all of the characters, a range of characters, or a single character from the file.

If you select to read a range of characters or only a single character, you again are asked to supply the limits of the range or the number of the character. If any of the numbers are outside the legal range, or if the lower limit exceeds the upper limit, an error message is displayed and no action takes place. If the program cannot open the file or read its contents, an error message to that effect is displayed. Check the spelling of the filename.

As mentioned above, if you intend to create a font by building it up through successive Read commands on a group of character definition files, invoke the Initialize command before you start. If not, the newly built definition may redefine character 0 to a blank or possibly crash the program, system, and printer. In addition, you may find it impossible to Read the uninitialized font definition after it has been stored on disk.

The Write command is used to store the contents of the character definition buffer into a file on the disk. You will be asked to enter a name for the file. If the file already exists, its contents are lost when the new file overwrites it. If the program cannot create the file or detects some I/O error, an error message is displayed. It is a good idea to make sure you have enough space on the disk for all the files you intend to create *before* you start the program. If the program crashes because of insufficient disk space, all of your work since the last Write command will be lost.

The Write command saves the entire character definition buffer. It cannot operate with only a range of characters or a single character.

The Quit command is used to exit the program. Remember to save any newly created or altered definitions with the Write command before Quitting or you will lose all your work.

Using the Program

Before you start the program, be sure that the switch in the printer that determines how the 2K buffer will be used is set so you can store your character definitions. To start the program, type fx80char followed by a carriage return. You can also enter a filename following the program name; fx80char will assume that the file is a character definition file and attempt to read it into the program's character buffer.

The usual sequence of actions during a session with fx80char involves Activating the printer's RAM character set, Reading a character definition, Editing some of the characters, Overlaying the new characters in the printer, and Printing the new character set. You usually repeat the Edit-Overlay-Print cycle several times during a session as you make changes to the character definitions. When you are finished, you probably will want to Write your revised character definitions to a file for permanent storage.

Circle **no. 52** on reader service card.

When you want to use a different character set from another program (for example, to print a draft copy of a document), run fx80char, Read the new character set from a file, then Overlay the new character set. After you exit from fx80char, the altered character set will be used for printing until you turn off the power or run fx80char again and replace the set.

Some of my friends and I have been using this program for some time. We have yet to find any errors. The most common mistake we make seems to be Quitting the program before Writing revised definitions to a file.

The Program

The source text for the program is divided among five files written in the Eco-C version of C for CP/M. These consist of the header file FX80.H (Listing One) and the files FX80CHAR.C, ACTIONS.C, EDCHAR.C, and PRNOUT.C (Listings Two through Five, respectively).

FX80.H (Listing One, page 34) contains the global #define, data structure, and variable declarations. This file should be included in all C source modules making up the program. In addition, *one and only one* of the C source modules must define the macro MAIN before inclusion of FX80.H. This defines the global character buffer pointer, struct chrary *chrs, in that file. All other files (those without the macro MAIN) will compile only a declaration of *chrs as an external variable.

struct chrary defines the structure of the elements that will make up the program's global character buffer. It is fairly straightforward. The chrnum member contains the number of the character, which corresponds to the numbers requested in the Edit, Overlay, and Read commands. The attrib element contains the attribute byte for the character as currently defined: the program uses only two values for this byte to distinguish between printing with the upper or lower eight pins of the print head. The program generates no spacing data for proportional characters, and ary simply contains the sequence of bytes needed to define the character to the printer.

The variable chrs is a pointer to a variable of type struct chrary. When the program starts, an array large enough to hold NUMCHARS elements of type struct chrary is allocated; chrs points to the first element. The variable then is used globally throughout the program.

FX80CHAR.C (Listing Two, page 36) contains the main program function and miscellaneous utility routines. The function main( ) attempts to allocate space for the global character buffer and to assign the variable *chrs to point to it. If the allocation fails, the program will abort. If the allocation succeeds, the program will zero (not initialize) the character buffer. The character buffer is allocated dynamically to decrease the size of the code file (and to speed up program assembly and linkage).

The program then checks to see if a command line argument has been specified. If so, the program interprets the argument as the name of a character definition file and attempts to read it into the global character buffer. When all initialization is complete, the function mainmenu( ) is invoked to handle the command loop.

ACTIONS.C (Listing Three, page 42) contains most of the functions that actually implement the commands available from the main menu level. These are all pretty obvious, many simply sending sequences of commands to the printer.

EDCHAR.C (Listing Four, page 50) contains the character editor function. After the user enters the number of the character to be edited, the element is "unpacked" into the static external variable cary. This is a two-dimensional array with one element of type unsigned, corresponding roughly to each bit in a variable of type struct chrary. Unpacking the bits into this variable makes the editing process much simpler and faster because individual bits of the character definition need not be accessed. The character display and cursor movement are fairly obvious. At the end of the edit, the character is checked for errors in definition and repacked.

PRNOUT.C (Listing Five, page 56) contains the function to print the character set. printall( ) loops through all 256 characters, testing whether the character falls into one of five classes. If it is a normal printable character or its italic counterpart, the character simply is sent to the printer. If it is a high-order control character or something other than a special low-order control character, the appropriate series of commands is sent to the printer. If the loop counter corresponds to character 127, a space is skipped. If it is a special character, as determined by the function isspecial( ), the pspecial( ) function is called. (Special characters correspond to things like the escape and carriage return.) This function basically does a brute force translation from the character number to the commands necessary to print a graphic version of the character.

External Functions

fx80char uses several external routines that are not members of the standard library. With a few exceptions, these are used to control the video properties of the terminal upon which the program is running.

The video control functions are from my C function library. Most programmers have a similar library containing routines to access functions available on their particular terminals. The functions revon( ) and revoff( ) are used to turn on and off the reverse video mode of the terminal. The gotoxy(column, line) function is used to position the cursor at an arbitrary location on the screen. clrline(line) clears the entire line that is specified as the function's argument. The function clreos(column, line) clears the screen, from the requested coordinates to the end of the screen, and leaves the cursor at the specified coordinates. Similarly, clrscrn( ) clears the entire screen and leaves the cursor at the home position.

My video routines assume that the home position has coordinates of (0, 0). Column numbers increase to the right and line numbers increase as the cursor moves down the screen. The program assumes it is running on a terminal with 24 lines of 80 col-

AZTEC The Most Powerful C

**for the IBM AT • MACINTOSH • MS DOS • CP/M-80 • ROM APPLICATIONS
IBM PC/XT • APPLE // • CP/M-86 • TRSDOS • CROSS DEVELOPMENT**

Why Professionals Choose Aztec C

AZTEC C compilers generate fast, compact code. AZTEC C is a sophisticated development system with assemblers, debuggers, linkers, editors, utilities and extensive run time libraries. AZTEC C is documented in detail. AZTEC C is the most accurate and portable implementation of C for microcomputers. AZTEC C supports specialized professional needs such as cross development and ROM code development. MANX provides qualified technical support.

AZTEC C86/PRO

— for the IBM AT and PC/XT

AZTEC C86/PRO provides the power, portability, and professional features you need to develop sophisticated software for PC DOS, MS DOS AND CP/M-86 based microsystems. The system also supports the generation of ROM based software for 8088/8086, 80186, and 80286 processors. Options exist to cross develop ROM code for 65xx, 8080, 8085, and Z80 processors. Cross development systems are also available that target most micro computers. Call for information on AZTEC C86/PRO support for XENIX and TOPVIEW.

POWERFUL — AZTEC C86/PRO 3.2 outperforms Lattice 2.1 on the DHRYSTONE benchmark 2 to 1 for speed (17.8 secs vs 37.1) while using 65% less memory (5.8k vs 14k). The AZTEC C86/PRO system also compiles in 10% to 60% less time and supports fast, high volume I/O.

PORTABLE — MANX Software Systems provides **real** portability with a family of compatible AZTEC C software development systems for PC DOS, MS DOS, CP/M-86, Macintosh, CP/M-80, APPLE //+, //e, and //c (NIBBLE - 4 apple rating), TRSDOS (80-MICRO - 5 star rating), and Commodore C64 (the C64 system is only available as a cross compiler - call for details). AZTEC C86/PRO is compatible with UNIX and XENIX.

PROFESSIONAL — For professional features AZTEC C86/PRO is unparalleled.

- Full C Compiler (8088/8086 - 80186 - 80286)
- Macro Assembler for 8088/8086/80186/80286
- Linkage Editor with ROM support and overlays
- Run Time Libraries - object libraries + source DOS 1.x; DOS 2.x; DOS 3.x; screen I/O; Graphics; UNIX I/O; STRING; simulated float; 8087 support; MATH; ROM; CP/M-86
- Selection of 8088/8086, 80186, or 80286 code generation to guarantee best choice for performance and compatibility
- Utility to convert AZTEC object code or libraries to Microsoft format. (Assembly + conversion takes less than half the time as Microsoft's MASM to produce MS object)
- Large memory models and sophisticated memory management
- Support products for graphics, DB, Screen, & ...
- ROMable code + ROM support + separate code and data + INTEL Hex Converter
- Symbolic Debugger & Other Utilities
- Full Screen Editor (like Vi)
- CROSS Compilers are available to APPLE //, Macintosh, CP/M-80, TRSDOS, COMMODORE C64, and ROM based 65xx, and 8080/8085/Z80
- Detailed Documentation

AZTEC C86/PRO-AT $500
(configured for IBM AT - options for 8088/8086)

AZTEC C86/PRO-PC/XT $500
(configured for IBM PC/XT - options for 80186/80286)

AZTEC C86/BAS includes C compiler (small model only), 8086 MACRO assembler, overlay linker, UNIX, MATH, SCREEN, and GRAPHICS libraries, debugger, and editor.

AZTEC C86/BAS $199
AZTEC C86/BAS (CP/M-86) $199
AZTEC C86/BAS (DOS + CP/M-86) $299
UPGRADE to AZTEC C86/PRO $310
C-TREE Database with source $399
C-TREE Database (object) $149

CROSS COMPILERS

Cross Compilers for ROM, MS DOS, PC DOS, or CP/M-86 applications.

VAX - > 8086/80xxx cross $5000
PDP-11 - > 8086/80xxx cross $2000

Cross Compilers with PC DOS or CP/M-86 hosts are $750 for the first target and $500 for each additional target. Targets: 65xx; CP/M-80; C64; 8080/8085/Z80; Macintosh; TRSDOS; 8086/8088/80186/80286; APPLE //.

AZTEC C68K

— for the Macintosh

For power, portability, and professional features AZTEC C68K-c is the finest C software development system available for the Macintosh.

The AZTEC C68K-c system includes a 68000 macro assembler, a linkage editor, a source editor, a mouse based editor, a SHELL development environment, a library of UNIX I/O and utility routines, full access and support of the Macintosh TOOLBOX routines, debugging aides, utilities, make, diff, grep, TTY simulator with upload & download (source supplied), a RAM disk (for 512K Mac), a resource maker, and a **no royalty** license agreement. Programming examples are included. (Over 600 pages of documentation).

AZTEC C68K-c requires a 128K Macintosh, and two disk drives (frugal developers can make do with one drive). AZTEC C68K supports the 512K Macintosh and hard disks.

AZTEC C68K-c (commercial system) $500
AZTEC C68K-p (personal system) $199
AZTEC C68K-p to AZTEC C68K-c upgrade $310
Mac C-tree database $149
Mac C-tree database with source $399
Lisa Kit (Pascal to AZTEC C68k object converter) .. $ 99

AZTEC C65

— for the APPLE //

"...The AZTEC C-system is one of the finest software packages I have seen..." NIBBLE review, July 1984.

The only commercial C development system available that runs native on the APPLE II+, IIc, and IIe, the AZTEC C65 development system includes a full floating point C compiler compatible with UNIX C and other MANX AZTEC C compilers, a 6502 relocating assembler, a linkage editor, a library utility, a SHELL development environment, a full screen editor, UNIX I/O and utility subroutines, simple graphics, and screen functions.

AZTEC C65 (Apple DOS 3.3) $199
AZTEC C65/PRO (Apple DOS + ProDos) $350
(call for availability)

AZTEC C II/PRO

— for CP/M-80

The first member of the AZTEC C family was the CP/M-80 AZTEC C compiler. It is "the standard" compiler for development on CP/M-80. The system includes the AZTEC C II C compiler, an 8080 assembler, a linkage editor, an object librarian, a full library of UNIX I/O and utility routines, CP/M-80 run time routines, the SMALL library (creates modules less than 3K in size), the fast linker for reduced development times, the ROM library, RMAC and M80 support, library source, support for DRI's SID/ZSID symbolic debugger, and more.

AZTEC C II/PRO $349
AZTEC CII/BAS $199
C-TREE Database with source $399
C-TREE Database in AZTEC object form $149

AZTEC C80

— for TRSDOS (Radio Shack Model III & 4)

"I've had a lot of experience with different C compilers, but the Aztec C80 Compiler and Professional Development System is the best I've seen." 80-Micro, December, 1984, John B. Harrell III

This sytem has most of the features of AZTEC C II for CP/M. It is perhaps the best software development system for the Radio Shack Model III and IV.

AZTEC C80 model 3 (no floating point) $149
AZTEC C80 model 4 (full) $199
AZTEC C80/PRO (full for model 3 and 4) $299

To order or for information call:

800-221-0440

(201) 530-7997 (NJ and outside U.S.A.). Or write: MANX SOFTWARE SYSTEMS, P.O. Box 55, Shrewsbury, N.J. 07701.

SHIPPING INFORMATION - Standard U.S. shipment is UPS ground (no fee). In the U.S. one day shipment is $20, two days is $10. Canadian shipment is $10. Two days shipment outside the U.S. is by courier and is freight collect.

MANX

TRS 80 RADIO SHACK TRS DOS is a trademark of TANDY.
APPLE DOS MACINTOSH is a trademark of APPLE.

For Technical Support
(Bug Busters) call: 201-530-6557

Circle no. 62 on reader service card.

umns each.

The program also uses some functions from the Eco-C standard library that are not available in the Unix C library. The _bdos( ) function, which provides direct access to the CP/M BDOS, is used for raw input from the keyboard and output to the printer. Similar functions are available in most CP/M-based versions of C. The setmem( ) function quickly initializes a block of memory to a specified value.

One additional item is worth noting here. The version of fopen( ) used in the listings does not use standard parameters. Unix C terminates lines with a single newline character, but CP/M uses the carriage return-line feed (CR/LF) pair of characters. Most CP/M-based implementations of C provide an automatic translation of CR/LF to LF when reading text from a disk. Conversely, single LF characters are translated to the CR/LF sequence when writing a text file. In Eco-C, the mode argument is declared to be "binary" by appending the character "b" to the standard mode (see functions readfl( ) and writefile( ) in Listing Three). This convention is used when no special handling of CR/LF sequences is desired.

Conclusions

If you are looking for things to improve, a couple of places in the program are ripe for diddling. First, the definition of struct chrary is rather wasteful. Declaring the attrib and ary fields as char could save about half the space now required. Another area that could be improved is the character-packing function, packit( ), in the file EDCHAR.C (Listing Four). As currently implemented, the program will not create characters that can be proportionally printed. This is due to the way packit( ) assigns the attribute byte.

It would also be simple to make the program a little more forgiving of user errors. You could have the program prompt users if they try to Quit before Writing their changes to disk. Another good idea might be to automatically Initialize the character buffer when it is allocated.

The program as it appears in the listings does not use any printf( )s. This made the program about 1.5K smaller at the expense of a couple of clumsy statement sequences. If you make changes that include calls to printf( ), you also might want to clean up some of those statements.

This program has made creating and printing alternate characters for the FX-80 much easier. You can change characters and immediately see the results, which is much faster and less frustrating than writing a program to send a fixed series of commands to the printer.

DDJ

fx80char (Text begins on page 28)

Listing One

```
/*
**      fx80.h -- global header file for fx80 character font editor
*/

#ifndef VOID
#define VOID            int
#endif

#define REVISION        "7 April 1984"
#define TITLE           "Character Font Editor -- FX-80 Vers. 1.0"

#define STATUSLINE      0               /* status line line number */
#define MENULINE        4               /* line at which menus start */
#define PROMPTLINE      20              /* line for prompts */
#define NAMELENGTH      16              /* max length of file name */

#define WIDTH           11              /* width of character cell */
#define HEIGHT          9               /* height of character cell */
#define NUMCHARS        256             /* number of possible characters */

#define BOXX            (unsigned) 30   /* left edge of character box */
#define BOXY            (unsigned) 5    /* top of character box */
#define YOFF            (BOXY + HEIGHT + 1)

#define BELL            7
#define ESC             27
#define CTRLE           5
#define CTRLX           24
#define CTRLS           19
#define CTRLD           4
#define CTRLF           6
#define CTRLQ           17
```

(Continued on page 36)

Listing One

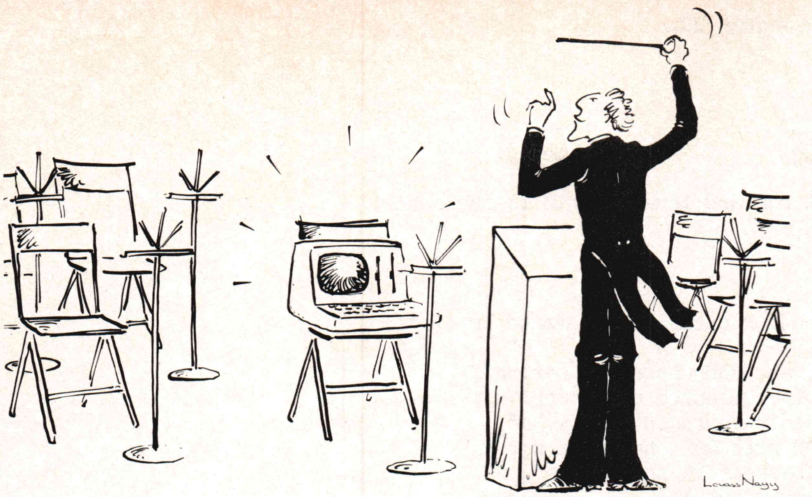
```
struct  chrary
        {
                unsigned        int     chrnum;
                unsigned        int     attrib;
                unsigned        int     ary[WIDTH];
        };

#define ELSIZE          (sizeof(struct chrary))
#define ARRAYSIZE       (NUMCHARS*ELSIZE)


#ifdef  MAIN            /* being included in main program file */

struct  chrary  *chrs;  /* pointer to global character array */

#else                   /* being included in subsidiary function file */

extern  struct  chrary  *chrs;

#endif
```

End Listing One

Listing Two

```
/*
**      fx80char.c -- character font editor for Epson FX80 printers.
**
**      This program can be used to create and alter character fonts
**  for downloading to Epson FX80 and FX100 printers.
**
**      David D. Clark
**      7 April 1984
*/

#include        <stdio.h>

#define         MAIN            /* notify the fx80.h header that this is
                                   the main program file */
#include        <fx80.h>


main(argc, argv)
int     argc;
char    *argv[];
{

        /* allocate character buffer area */
        if ((chrs = (struct chrary *) calloc(NUMCHARS, ELSIZE)) == NULL)
        {
                fputs("No room to allocate edit buffer.\n", stderr);
                exit(1);
        }

        /* try to read in an initial font file if requested */
        if (argc > 1)
        {
                readfl(argv[1], 0, NUMCHARS);
        }

        mainmenu();
        clrscrn();
        exit(0);
}
```

(Continued on page 38)



Would you hire an entire band when all you need is one instrument? Of course not.

So why use a whole orchestra of computers when all you need is one to develop software for virtually any type of micro-processor?

The secret? Avocet's family of cross-assemblers. With Avocet cross-assemblers you can develop software for practically every kind of processor — *without having to switch to another development system along the way!*

Cross-Assemblers to Beat the Band!

Development Tools That Work

Avocet cross-assemblers are fast, reliable and user-proven in over 4 years of actual use. Ask NASA, IBM, Xerox or the hundreds of other organizations that use them. Every time you see a new micro-processor-based product, there's a good chance it was developed with Avocet cross-assemblers.

Avocet cross-assemblers are easy to use. They run on almost any personal computer and process assembly language for the most popular microprocessor families.

Your Computer Can Be A Complete Development System

Avocet has the tools you need to enter and assemble your soft-ware and finally cast it in EPROM:

VEDIT Text Editor makes source code entry a snap. Full-screen editing plus a TECO-like command mode for advanced tasks. Easy installation - INSTALL program supports over 40 terminals and personal computers. Customizable keyboard layout. CP/M-80, CP/M-86, MSDOS, PCDOS $150

EPROM Programmers let you program, verify, compare, read, display EPROMS but cost less because they communicate through your personal computer or terminal. No personality modules! On-board intelligence provides menu-based setup for 34 different EPROMS, EEPROMS and MPUs (40-pin devices require socket adaptors). Self-contained unit with internal power supply, RS-232 interface, Textool ZIF socket. Driver software (sold separately) gives you access to all programmer features through your computer, lets you download cross-assembler output files, copy EPROM to disk.

Model 7228 Advanced Programmer — Supports all PROM types listed. Super-fast "adaptive" programming algorithm programs 2764 in 1.1 minutes.

Model 7128 Standard Programmer — Lower-cost version of 7228. Supports all PROM types except "A" versions of 2764 and 27128. Standard programming algorithm programs 2764 in 6.8 minutes.

Avocet Cross-assembler		Target Microprocessor	CP/M-80	CP/M-86 IBM PC, MSDOS**
XASM04	*NEW*	6804	$ 250.00	$ 250.00
XASM05		6805	200.00	250.00
XASM09		6809	200.00	250.00
XASM18		1802/1805	200.00	250.00
XASM48		8048/8041	200.00	250.00
XASM51		8051	200.00	250.00
XASM65		6502/65C02	200.00	250.00
XASM68		6800/01, 6301	200.00	250.00
XASM75		NEC 7500	500.00	500.00
XASM85		8085	250.00	250.00
XASM400		COP400	300.00	300.00
XASMF8		F8/3870	300.00	300.00
XASMZ8		Z8	200.00	250.00
XASMZ80		Z80	250.00	250.00
XMAC682	*NEW*	68200	595.00	595.00
XMAC68K	*NEW*	68000/68010	595.00	595.00

Model 7956 and 7956-SA Gang Programmers — Similar features to 7228, but program as many as 8 EPROMS at once. 7956-SA stand-alone version copies from a master EPROM. 7956 lab version has all features of stand-alone plus RS-232 interface.

EPROM: 2758, 2716, 2732, 2732A, 2764, 2764A, 27128, 27128A, 27256, 2508, 2516, 2532, 2564, 68764, 68766, 5133, 5143. **CMOS:** 27C16, 27C32, 27C64, MC6716. **EEPROM:** 5213, X2816A, 48016, I2816A, 5213H. **MPU (w/adaptor):** 8748, 8748H, 8749, 8749H, 8741, 8742, 8751, 8755.

7228	**Advanced Programmer**	**$ 549**
7128	**Standard Programmer**	**429**
7956	**Laboratory Gang Programmer**	**1099**
7956-SA	**Stand-Alone Gang Programmer**	**879**
GDX	**Driver Software**	**95**
481	**8748 Family Socket Adaptor**	**98**
511	**8751 Socket Adaptor**	**174**
755	**8755 Socket Adaptor**	**135**
CABLE	**RS-232 Cable (specify gender)**	**30**

HEXTRAN Universal HEX File Converter — Convert assembler output to other formats for downloading to development systems and target boards. Also useful for examining object file, changing load addresses, extracting parts of files. Converts to and from Intel, Motorola, MOS, RCA, Fairchild, Tektronix, TI, Binary and HEX/ASCII Dump formats. For CP/M, CP/M-86, MSDOS, PCDOS $250

Ask about UNIX.

68000 CROSS-ASSEMBLER — With exhaustive field testing completed, our 68000 assembler is available for immediate shipment. XMAC68K supports Motorola standard assembly language for the 68000 and 68010. Macros, cross-reference, structured assembly statements, instruction optimization and more. Linker and librarian included. Comprehensive, well-written manual.

To find out more, call us toll-free.

1-800-448-8500

(in the U.S. Except Alaska and Hawaii)

VISA and Mastercard accepted. All popular disc formats now available --please specify. Prices do not include shipping and handling -- call for exact quotes. OEM INQUIRIES INVITED.

*Trademark of Digital Research **Trademark of Microsoft

AVOCET SYSTEMS INC.™

Sales and Development:
10 Summer Street
P.O. Box 490, Dept. 485-DDJ
Rockport, Maine 04856
(207) 236-9055 Telex: 467210 AVOCET CI

Corporate Offices:
804 South State Street
Dover, Delaware 19901

Listing Two

```
/*
**      mainmenu -- outer most command level.  This is the actual

**  command p ocessing loop.
*/

char    *mnul[] =       /* menu for outer-most command level */
        {
          " -- Activate/deactivate RAM character set.\n",
          " -- Edit the currently selected character.\n",
          " -- Initialize the edit character array.\n",
          " -- Load printer RAM with ROM character set.\n",
          " -- Overlay new characters in printer.\n",
          " -- Print the character set.\n",
          " -- Read character file.\n",
          " -- Write character file.\n",
          " -- Quit.\n"
        };

char    *cmdl =         /* acceptable user responses to this level */
        {

            "AEILOPRWQ"
        };

int     ncmdl = 9;      /* number of acceptable commands */

VOID    mainmenu()
{
int     cmd;            /* command character */

        while (TRUE)    /* loop forever */
        {
                clrscrn();
                shortstat();
                cmd = domenu(mnul, cmdl, ncmdl);
                switch (cmd)
                {
                        case 'A':
                                activate();
                                break;

                        case 'E':
                                edchar();
                                break;

                        case 'I':
                                inichars();
                                break;

                        case 'L':
                                loadram();
                                break;

                        case 'O':
                                overlay();
                                break;

                        case 'P':
                                printall();
                                break;

                        case 'R':
                                readfile();
                                break;

                        case 'W':
                                writefile();
                                break;
```

```
				case 'Q':
					return;

			}
		}
}


/*
**	The following functions are utilities used throughout the program.
*/

/*
**	domenu -- print out specified menu and get an acceptable response.
*/

int	domenu(mnu, cmds, ncmds)
char	*mnu[],
	*cmds;
int	ncmds;
{
int	c;
unsigned	i;

		fputs("\n\n\n", stdout);
		for (i = 0; i < ncmds; i++)
		{
			fputs("\t\t", stdout);
			revon();
			putc(cmds[i], stdout);
			revoff();
			fputs(mnu[i], stdout);
		}

		do
		{
			c = (islower(c = mygetc()) ? toupper(c) : c);
			if (index(cmds, c) == NULL)
			{
				c = NULL;
				putc(BELL, stdout);
			}
		} while (c == NULL);

		return (c);
}


/*
**	selchar -- select a character by number.
*/

unsigned	selchar()
{
char	charstr[6];

	gotoxy(0, PROMPTLINE);
	fputs("character number?  ", stdout);
	if (fgets(charstr, 5, stdin) == NULL)
		return (32);	/* have to return something */
	return ((unsigned) atoi(charstr));
}


/*
**	pputc -- put a character to the printer.
*/

VOID	pputc(c)
int	c;
{
	_bdos(5, c);		/* send char directly to printer */
}
```

(Continued on next page)

Listing Two

```
/*
**      mygetc -- get raw keyboard input.
*/

int     mygetc()
{
int     c;

        while (!(c = _bdos(6, 0x00ff)))
                ;                               /* wait for a character */
        return (c);
}


/*
**      pause -- wait for the user to type a character.
*/

VOID    pause()
{
        fputs("Type any character to continue.  ", stdout);
        mygetc();
        putc('\n', stdout);
}


/*
**      cantopen -- print error message when unable to open "file".
*/

VOID    cantopen(file)
char    *file;
{
                fputs("Can't open ", stderr);
                fputs(file, stderr);
               putc('\n', stderr);
                pause();
}


/*
**      filerr -- print error message after file i/o error.
*/

VOID    filerr()
{
        fputs("\nFile i/o error\n", stderr);
        pause();
}


/*
**      shortstat -- print short status line.
*/

VOID    shortstat()
{
        clrline(STATUSLINE);
        fputs("\t\t", stdout);
        revon();
        fputs(TITLE, stdout);
        revoff();
}
```

(Continued on page 42)

ALL SALES ARE MADE SUBJECT TO THE TERMS OF OUR 90 DAY LIMITED WARRANTY. A COPY OF THIS WARRANTY IS AVAILABLE FREE, ON REQUEST.

DIGITAL RESEARCH COMPUTERS

(214) 225-2309

DISK DRIVE SPECIAL!

5¼" Double Sided-Double Density

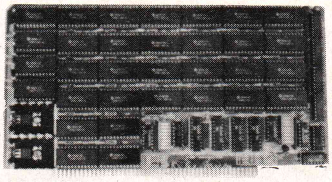
Brand New, Unused. Mfg. in Japan by Canon. 2/3 Height - Direct Drive! Industry Standard Pin Out. 6MS Access.

$49.95 ea — **2 FOR $85**

Sold on a First Come Basis
Add $2 Each U.P.S.

64K S100 STATIC RAM

$159.00 KIT



NEW!
LOW POWER!
150 NS ADD $10

BLANK PC BOARD WITH DOCUMENTATION
$49.95

SUPPORT ICs + CAPS
$17.50

FULL SOCKET SET
$14.50

FULLY SUPPORTS THE NEW IEEE 696 S100 STANDARD (AS PROPOSED)

FOR 56K KIT $145

ASSEMBLED AND TESTED ADD $50

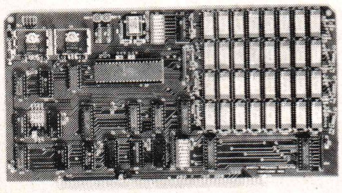
PRICE CUT!

FEATURES:
* Uses new 2K x 8 (TMM 2016 or HM 6116) RAMs.
* Fully supports IEEE 696 24 BIT Extended Addressing.
* 64K draws only approximately 500 MA.
* 200 NS RAMs are standard. (TOSHIBA makes TMM 2016s as fast as 100 NS. FOR YOUR HIGH SPEED APPLICATIONS.)
* SUPPORTS PHANTOM (BOTH LOWER 32K AND ENTIRE BOARD).
* 2716 EPROMs may be installed in any of top 48K.
* Any of the top 8K (E000 H AND ABOVE) may be disabled to provide windows to eliminate any possible conflicts with your system monitor, disk controller, etc.
* Perfect for small systems since BOTH RAM and EPROM may co-exist on the same board.
* BOARD may be partially populated as 56K.

256K S-100 SOLID STATE DISK SIMULATOR!

WE CALL THIS BOARD THE "LIGHT-SPEED-100" BECAUSE IT OFFERS AN ASTOUNDING INCREASE IN YOUR COMPUTER'S PERFORMANCE WHEN COMPARED TO A MECHANICAL FLOPPY DISK DRIVE.

PRICE CUT!



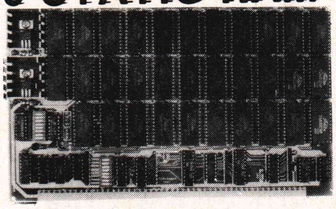
FEATURES:
* 256K on board, using + 5V 64K DRAMS.
* Uses new Intel 8203-1 LSI Memory Controller.
* Requires only 4 Dip Switch Selectable I/O Ports.
* Runs on 8080 or Z80 S100 machines.
* Up to 8 LS-100 boards can be run together for 2 Meg. of On Line Solid State Disk Storage.
* Provisions for Battery back-up.
* Software to mate the LS-100 to your CP/M* 2.2 DOS is supplied.
* The LS-100 provides an increase in speed of up to 7 to 10 times on Disk Intensive Software.
* Compare our price! You could pay up to 3 times as much for similar boards.

BLANK PCB (WITH CP/M* 2.2 PATCHES AND INSTALL PROGRAM ON DISKETTE)
$69.95
(8203-1 INTEL $29.95)

$199.00
#LS-100 (FULL 256K KIT)

64K SS-50 STATIC RAM

$129.95 (48K KIT)



NEW!
LOW POWER!
RAM OR EPROM!

BLANK PC BOARD WITH DOCUMENTATION
$52

SUPPORT ICs + CAPS
$18.00

FULL SOCKET SET
$15.00

56K Kit $139
64K Kit $169

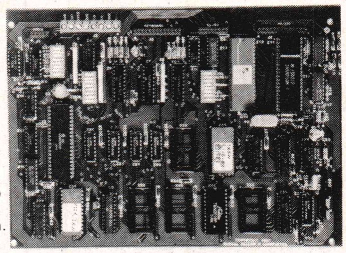
ASSEMBLED AND TESTED ADD $50

FEATURES:
* Uses new 2K x 8 (TMM 2016 or HM 6116) RAMs.
* Fully supports Extended Addressing.
* 64K draws only approximately 500 MA.
* 200 NS RAMs are standard. (TOSHIBA makes TMM 2016s as fast as 100 NS. FOR YOUR HIGH SPEED APPLICATIONS.)
* Board is configured as 3-16K blocks and 8-2K blocks (within any 64K block) for maximum flexibility.
* 2716 EPROMs may be installed anywhere on Board.
* Top 16K may be disabled in 2K blocks to avoid any I/O conflicts.
* One Board supports both RAM and EPROM.
* RAM supports 2MHZ operation at no extra charge!
* Board may be partially populated in 16K increments.

THE NEW ZRT-80 CRT TERMINAL BOARD!

A LOW COST Z-80 BASED SINGLE BOARD THAT ONLY NEEDS AN ASCII KEYBOARD, POWER SUPPLY, AND VIDEO MONITOR TO MAKE A COMPLETE CRT TERMINAL. USE AS A COMPUTER CONSOLE, OR WITH A MODEM FOR USE WITH ANY OF THE PHONE-LINE COMPUTER SERVICES.

FEATURES:
* Uses a Z80A and 6845 CRT Controller for powerful video capabilities.
* RS232 at 16 BAUD Rates from 75 to 19,200.
* 24 x 80 standard format (60 Hz).
* Optional formats from 24 x 80 (50 Hz) to 64 lines x 96 characters (60 Hz).
* Higher density formats require up to 3 additional 2K x 8 6116 RAMS.
* Uses N.S. INS 8250 BAUD Rate Gen. and USART combo IC.
* 3 Terminal Emulation Modes which are Dip Switch selectable. These include the LSI-ADM3A, the Heath H-19, and the Beehive.
* Composite or Split Video.
* Any polarity of video or sync.
* Inverse Video Capability.
* Small Size: 6.5 x 9 inches.
* Upper & lower case with descenders.
* 7 x 9 Character Matrix.
* Requires Par. ASCII keyboard.



BLANK PCB WITH 2716 CHAR. ROM, 2732 MON. ROM
$49.95

SOURCE DISKETTE - ADD $10
SET OF 2 CRYSTALS - ADD $7.50

WITH 8 IN. SOURCE DISK! (CP/M COMPATIBLE)

$99.95
ZRT-80 (COMPLETE KIT, 2K VIDEO RAM)

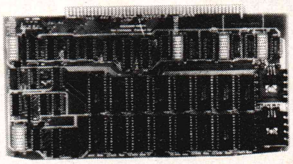
32K S100 EPROM/STATIC RAM

FOUR FUNCTION BOARD!

NEW! NEW!

EPROM II FULL EPROM KIT
$69.95

A&T EPROM ADD $35.00



BLANK PC BOARD WITH DATA
$39.95

SUPPORT IC'S PLUS CAPS
$16

FULL SOCKET SET
$15

We took our very popular 32K S100 EPROM Card and added additional logic to create a more versatile EPROM/RAM Board.

FEATURES:
* This one board can be used in any one of four ways:
  A. As a 32K 2716 EPROM Board
  B. As a 32K 2732 EPROM Board (Using Every Other Socket)
  C. As a mixed 32K 2716 EPROM/2K x 8 RAM Board
  D. As a 32K Static RAM Board
* Uses New 2K x 8 (TMM2016 or HM6116) RAM's
* Fully Supports IEEE 696 Buss Standard (As Proposed)
* Supports 24 Bit Extended Adressing
* 200 NS (FAST!) RAM'S are standard on the RAM Kit
* Supports both Cromemco and North Star Bank Select
* Supports Phantom
* On Board wait State Generator
* Every 2K Block may be disabled
* Addressed as two separate 16K Blocks on any 64K Boundary
* Perfect for MP/M* Systems
* RAM Kit is very low power (300 MA typical)

PRICES SLASHED!

32K STATIC RAM KIT — $109.95
For RAM Kit A&T — Add $40

Digital Research Computers

P.O. BOX 461565 • GARLAND, TEXAS 75046 • (214) 225-2309

TERMS: Add $3.00 postage. We pay balance. Orders under $15 add 75¢ handling. No. C.O.D. We accept Visa and MasterCharge. Tex. Res. add 5-1/8% Tax. Foreign orders (except Canada) add 20% P & H. Orders over $50, add 85¢ for for insurance.

*TM OF DIGITAL RESEARCH INC. (CALIF.)

WE ARE NOT ASSOCIATED WITH DIGITAL RESEARCH INC. (CALIF.) THE SUPPLIERS OF CPM SOFTWARE

Circle no. 33 on reader service card.

Listing Two

```
/*
**      longstat -- print a long status line.
*/

VOID    longstat(charnum)
int     charnum;
{
char    curchar[6];             /* ASCII representation of charnum */

        clrline(STATUSLINE);
        revon();
        fputs(TITLE, stdout);
        gotoxy(42, STATUSLINE);
        fputs("Char. No. ", stdout);
        itoa(curchar, charnum);
        fputs(curchar, stdout);

        gotoxy(58, STATUSLINE);
        fputs("Char = ", stdout);
        revoff();

        if (isprint(charnum & 0x7f))
                if (charnum > 127)
                {
                        revon();
                        putc(charnum, stdout);
                        revoff();
                }
                else
                        putc(charnum, stdout);
        else
        {
                revon();
                fputs("Can't show", stdout);
                revoff();
        }
}
```

End Listing Two

Listing Three

```
/*
**      actions.c -- action routines for character editor
*/

#include        <stdio.h>
#include        <fx80.h>


/*
**      activate -- activate the RAM character set.
*/

static  ramact = FALSE;         /* assume ROM set active at start of program */

VOID    activate()
{
        gotoxy(0, PROMPTLINE);
        if (!ramact)
        {
```

```
                pputc(ESC);
                pputc('%');
                pputc(1);
                pputc(0);
                ramact = TRUE;
                fputs("RAM character set active.\n", stdout);
        }
        else
        {
                pputc(ESC);
                pputc('%');
                pputc(0);
                pputc(0);
                ramact = FALSE;
                fputs("ROM character set active.\n", stdout);
        }
        pause();
}


/*
**      inichars -- initialize the global character array.
*/

VOID    inichars()
{
unsigned        i;
struct  chrary  *ptr;

        gotoxy(0, PROMPTLINE);
        fputs("Initializing character array...\n", stdout);

        setmem(chrs, ARRAYSIZE, 0);
        for (i = 0, ptr = chrs; i < NUMCHARS; i++, ptr++)
                ptr->chrnum = i;

        pause();
}


/*

**      loadram -- load ROM character set into RAM.
*/

VOID    loadram()
{
        gotoxy(0, PROMPTLINE);
        pputc(ESC);
        pputc(':');
        pputc(0);
        pputc(0);
        pputc(0);
        fputs("Printer RAM loaded from ROM.\n", stdout);
        pause()
}


/* menu and responses for the overlay and read commands */

char    *mnu2[] =
        {
          " -- All characters.\n",
          " -- Range of characters.\n",
          " -- Single character.\n"
        };

char    *cmd2 =
        {
          "ARS"
        };
```

(Continued on next page)

Listing Three

```
int     ncmd2 = 3;


/*
**      overlay -- overlay the characters in the printer RAM with
**  the characters in chrs.
*/

VOID    overlay()
{
int     cmd;
unsigned        i, rlo, rhi;
struct  chrary  *myptr;

        clrscrn();
        shortstat();
        cmd = domenu(mnu2, cmd2, ncmd2);
        gotoxy(0, PROMPTLINE);
        switch (cmd)
        {
                case 'A':
                        fputs("Overlaying entire character set...\n", stdout);
                        for (i = 0, myptr = chrs; i < NUMCHARS; i++, myptr++)
                                defchar(myptr);
                        break;

                case 'R':
                        clreos(0, (PROMPTLINE - 1));
                        fputs("For the low end of the range,\n", stdout);
                        rlo = selchar();
                        clreos(0, (PROMPTLINE - 1));
                        fputs("For the high end of the range,\n", stdout);
                        rhi = selchar();
define:                 if (rlo >= 0 && rhi < NUMCHARS && rlo <= rhi)
                        {
                                myptr = chrs;
                                /* skip unwanted characters */
                                while (myptr->chrnum != rlo)
                                        myptr++;
                                for (i = rlo; i <= rhi; i++, myptr++)
                                        defchar(myptr);
                        }
                        else
                                fputs("Illegal character range.\n", stderr);
                        break;

                case 'S':
                        rlo = rhi = selchar();
                        goto define;
        }

        pause();
}


/*
**      defchar -- define a character to the printer.
*/

VOID    defchar(p)
struct  chrary  *p;
{
unsigned        j;

        pputc(ESC);
```

(Continued on page 48)

THE ANATOMY OF A

INTRODUCING Interface Technologies' Modula-2 Software Development System

The computer press is hailing Modula-2 as "the next standard in programming languages." Modula-2 combines the strengths of Pascal with the features that made C so popular, like independent compilation and direct hardware control.

But until today, no company offered a Modula-2 system that made the development of software fast, easy and efficient. Now, though, there's a new tool at your disposal.

The fast, powerful tool for programmers

The breakthrough is here: Interface Technologies' new Modula-2 Software Development System for the IBM® PC, XT, AT and compatible computers to give programmers the same quantum leap in productivity spreadsheets and word processors gave to end-users. It can reduce monotonous wait time, will dramatically increase speed, help stop thoughtless mistakes, and free you to become more creative in virtually all of your programming efforts.

How to speed input and eliminate 30% of errors

Thirty percent of programming mistakes are syntax errors and simple typos in the program structure. Our "syntax-directed" Modula-2 editor does away with these time-consuming headaches once and for all.

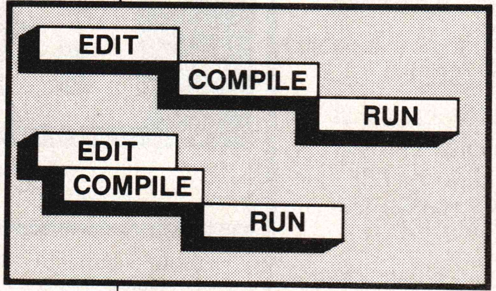
Enter complete statements with one keystroke.

It speeds input by cutting manual typing as much as 90%, letting you enter statements with a single keystroke. For example, if you type a capital "I" to begin a line, the editor completes the logical "IF THEN" statement automatically, so you can concentrate on what you want to program, rather than concentrate on what you're typing.

The editor locks out errors, finishing statements and procedures in perfect accord with the standardized rules of Modula-2. It also indents and formats your text automatically, making programs easy to read and maintain, an important feature on big projects.

And if you leave an undefined variable or data type, the editor detects the mistake and gives you the option of on-line "help" to correct it. No other programming text editor offers you so much innovation at any price.

How to turn "wait time" into "work time"



It not only has a faster compiler, it also saves time by compiling while you edit.

The vast majority of programming time is spent waiting, and the biggest slowdown is most often with compilers.

Our compiler turns wait time to work time with a new innovation that lets you compile in the "background."

With background compilation, your program is automatically compiled into object code line by line as you work, every minute you spend writing or editing a Modula-2 program!

When you're finished editing, all that's left for the compiler is a quick mopping up job that generates optimized native code in a single pass.

How quick is "quick"?

Thanks to background compilation and the fact that the compiler itself is so fast, Interface Technologies' compiler turns 100 lines of typical Modula-2 text into optimized machine code in *under five seconds.*

Plus the Interface compiler produces compact code with execution speed superior to that produced by any other Modula-2 compiler on the market.

How to do two things at once

Along with the background compiler and syntax-directed editor, which can save you hours every day and make you more productive, Interface Technologies' Software Development System gives your monitor

windows so you can refer to one file while you edit another simultaneously, saving you even more time.

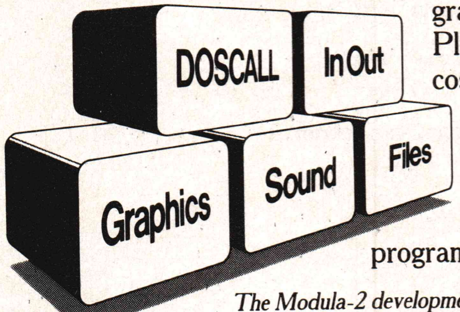
Concurrent editing of two or more files is especially useful when doing programming work that's intended for separate compilation, and Interface Technologies has the only Modula-2 system on the market that provides you with this helpful benefit for developing software.

Work with multiple files faster, easier in windows.

How preprogrammed modules speed development

One of the advantages of Modula-2 is that it lets you build large, reliable programs quickly, by linking together many smaller "building-block" modules.

The development system's toolkit of precompiled program modules includes the standard Modula-2 library, and adds exclusive link-and-run modules for direct calls to the operating system, sound, and color graphics support. Plus you get low-cost updates from the Interface Technologies fast-growing library of new programming modules.



The Modula-2 development system's toolkit of ready-made modules turns big programs into smaller projects.

Increase productivity for $249

Interface Technologies' Software Development System is fast, powerful and unlimited. It works so well that it's the same tool Interface Technologies is using to write business and consumer applications in Modula-2.

For $249, you get the syntax-directed editor and compiler, linker, module library and tutorial that will have even modestly experienced programmers writing in Modula-2 in days. And you have full rights to your work; there's no license fee for programs you develop with the Interface Technologies system.

You can use it on any IBM® PC, XT, AT or compatible computer with two double-sided, double-density floppy drives and 320K RAM diskette.

You get a thoroughly indexed, comprehensive user's manual and free telephone support from Interface Technologies. But the most important thing you get is the future, and *the programming language of the future is Modula-2*, and now it's easier than ever.

For more information, or to order the Modula-2 Software Development System, call 1-800-922-9049 today. In Texas, call (713) 523-8422.

You can also order or request further information by mail. Just fill out the coupon below and send it in. Act today and receive your system soon.

Operates on the new IBM PC AT, as well as the PC, XT, and all other IBM-compatible computers.

BREAKTHROUGH

NAME ____________________
ADDRESS ____________________
CITY ____________ STATE ______ ZIP ________
PHONE ____________________
PLEASE CHECK ONE:
☐ AMERICAN EXPRESS ☐ VISA ☐ MASTERCARD
☐ CHECK ENCLOSED
CHARGE ACCOUNT NUMBER
☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

EXPIRATION DATE SIGNATURE
PLEASE SEND ME ______ COPIES @ $249 EACH.
INTERFACE TECHNOLOGIES CORPORATION
3336 RICHMOND, SUITE 200, HOUSTON, TX 77098
Texas residents, add 6.125% Sales Tax. **DD/2**

INTERFACE TECHNOLOGIES

MODULA-2 SOFTWARE DEVELOPMENT SYSTEM

Listing Three

```
            pputc('&');
            pputc(0);
            pputc(p->chrnum);
            pputc(p->chrnum);

            pputc(p->attrib);
            for (j = 0; j < WIDTH; j++)
                    pputc(p->ary[j]);
    }


    /*
    **      readfile -- read a file of character definitions.
    */

    VOID    readfile()
    {
    char            name[NAMELENGTH];
    int             cmd;
    unsigned        rlo, rhi;

            gotoxy(0, PROMPTLINE);
            fputs("Read what character file?  ", stdout);
            fgets(name, NAMELENGTH, stdin);
            clrscrn();
            shortstat();
            cmd = domenu(mnu2, cmd2, ncmd2);
            gotoxy(0, PROMPTLINE);
            switch (cmd)
            {
                    case 'A':
                            readfl(name, 0, NUMCHARS);
                            break;

                    case 'R':
                            clreos(0, (PROMPTLINE - 1));
                            fputs("For the low end of the range,\n", stdout);
                            rlo = selchar();
                            clreos(0, (PROMPTLINE - 1));
                            fputs("For the high end of the range,\n", stdout);
                            rhi = selchar();
doread:                     if (rlo >= 0 && rhi < NUMCHARS && rlo <= rhi)
                                    readfl(name, rlo, rhi);
                            else
                                    fputs("Illegal character range.\n", stderr);
                            break;

                    case 'S':
                            rlo = rhi = selchar();
                            goto doread;
            }
    }


    /*
    **      readfl -- do the actual read of a character definition file.
    **  Only read characters from "lo" to "hi" into the global character
    **  buffer.
    */

    VOID    readfl(name, lo hi)
    char    *name;
    unsigned        lo, hi;
    {
    char    c, *ptr;
    unsigned        i;
    FILE    *fd;

            if ((fd = fopen(name, "rb")) == NULL)
```

```
        {
                cantopen(name);
                return;
        }

        fputs("Reading...", stdout);

        /* skip past unwanted characters */
        for (i = 0, ptr = (char *) chrs;
             (i < (lo*ELSIZE)) && ( c = getc(fd)) != EOF);
             i++, ptr++
            )
                ; /* do nothing */

        if (lo > 0 && i < (lo*ELSIZE - 1))
                filerr();

        /* now read the characters into the array */
        for (
             i = (lo*ELSIZE); /* ptr is already where it should be */
             (i < (hi*ELSIZE + ELSIZE)) && ((c = getc(fd)) != EOF);
             i++, ptr++
            )
                *ptr = c;

        if (hi > 0 && i < (hi*ELSIZE + ELSIZE - 1))
                filerr();

        fclose(fd);
}


/*
**      writefile -- write a file of character definitions.
*/

VOID    writefile()
{
char    c, *ptr;
char    name[NAMELENGTH];
unsigned        i;
struct  chrary  *myptr;
FILE    *fd;


        gotoxy(0, PROMPTLINE);
        fputs("Write characters to what file?  ", stdout);
        fgets(name, NAMELENGTH, stdin);
        putc('\n', stdout);
        if ((fd = fopen(name, "wb")) == NULL)
        {
                cantopen(name);
                return;
        }

        fputs("Writing...", stdout);
        for (i = 0, ptr = (char *) chrs; i < ARRAYSIZE; i++, ptr++)
        {
                if (putc(*ptr, fd) == EOF)
                {
                        filerr();
                        fclose(fd);
                        return;
                }
        }

        fclose(fd);
}
```

End Listing Three

(Listing four begins on next page)

Listing Four

```
/*
**      edchar.c -- character editing functions for the fx80
**   character editor program.
*/

#include        <stdio.h>
#include        <fx80.h>


/* unpacked version of the character being edited */
static  unsigned        cary[WIDTH][HEIGHT];


/*
**      edchar -- the main character editing function.
*/

VOID    edchar()
{
int     c;
unsigned        myx, myy, charnum;

        gotoxy(0, PROMPTLINE - 1);
        fputs("Edit", stdout);
        charnum = selchar();
        clrscrn();
        drwbox();
        longstat(charnum);
        unpack(charnum);
        flashit(charnum);
        myx = 1;
        myy = 1;
        boxgotoxy(myx, myy);

        do
        {
                switch (c = (islower(c = mygetc()) ? toupper(c) : c))
                {

                        case '8':               /* up */
                        case CTRLE:
                                if (myy < HEIGHT)
                                        boxgotoxy(myx, ++myy);
                                break;

                        case '2':               /* down */
                        case CTRLX:
                                if (myy > 1)
                                        boxgotoxy(myx, --myy);
                                break;

                        case '4':               /* left */
                        case CTRLS:
                                if (myx > 1)
                                        boxgotoxy(--myx, myy);
                                break;

                        case '6':               /* right */
                        case CTRLD:
                                if (myx < WIDTH)

                                        boxgotoxy(++myx, myy);
                                break;

                        case '5':               /* toggle pixel */
                        case CTRLF:
                                if (cary[myx - 1][myy - 1])
                                {
                                        if ((myx - 1)%2)
```

(Continued on page 52)

DEZIGN

THE STRUCTURED PROGRAMMING TOOL FOR MODERN TIMES

- Design your programs right on the screen, using modern techniques based on the popular Jackson Structured Programming method (JSP)!
- **DEZIGN** is more than just another flowcharting tool. It is an integrated tool for designing and documenting programs and for generating ADA, C, PASCAL, and PL/I source code, as well as dBASE II and dBASE III command files.
- **DEZIGN** enables you to create Data and Program Structure Diagrams using the Sequence, Selection (IF-THEN-ELSE), and Iteration (DO WHILE) constructs; assign detailed statements to the diagrams; and synthesize source code from the control logic represented on the diagrams and the detailed statements assigned to them.
- **DEZIGN** lists for $200. It runs on the IBM PC, XT, or AT and requires 128K RAM, one disk drive, and an 80-column color or monochrome display.
  - DEZIGN-PC runs under DOS 2.0, 2.1, and 3.0.
  - DEZIGN-86 runs under CP/M-86 1.1.
- Want to learn more? Please contact us concerning pricing and availability of JSP reference texts and seminars.

ZEDUCOMP • P.O. BOX 68 • STIRLING, NJ 07980

(201) 755-2262

dBASE II and dBASE III are trademarks of Ashton-Tate, Inc.

Circle **no. 126** on reader service card.

GO FORTH, UNIX! . . . with u4th

Are you interested in improving your software productivity? Do you have a **UNIX** system or engineering work station? Now you can realize the exceptional capability of moving your Forth applications to the world of **UNIX** and **XENIX**. Experience the productivity enhancement of an interactive programming environment, and still code in C when necessary.

UBIQUITOUS SYSTEMS announces **u4th**, the first Forth completely tailored for **UNIX**. **u4th** is a fast direct-threaded Forth written in portable C, yet capable of execution speeds comparable to many assembler Forths. Great for **AI** research and delivery.

Some features are:

- Access to standard **UNIX** system calls
- Ability to incorporate new primitives written in C
- Object-Oriented Forth included!
- Can pass commands through to **UNIX**

Binary License: Xenix $395.00 Plexus $895.00.
Call about others. OEM's: Special terms.

UBIQUITOUS SYSTEMS

13333 Bel-Red Road N.E. Bellevue, Wa. 98005

(206) 641-8030

9:00-noon Weekdays

UNIX(TM) AT&T XENIX(TM) MICROSOFT

Circle **no. 102** on reader service card.

DATESTAMPER™ has the answers

```
   Drive B1:   4 files,  using 15K (110K FREE)      14:01-0? Feb
  -- file        size        created           accessed          modified --

B1: ADDRES  .DAT   5K | 22:01-17 Jan     08:30-01 Feb      08:23-01 Feb
B1: JSMITH  .LTR   2K | 16:30-24 Dec'84  11:59-10 Feb      16:30-24 Dec'84
B1: TEST1   .BAS   4K | 09:34-22 Jan     16:27-30 Jan      09:35-22 Jan
B1: TEST2   .BAS   4K | 11:55-01 Feb                       11:55-01 Feb
```

When did we print that letter?

Has the mailing list been updated?

Which is the latest version?

DateStamper™ keeps your CP/M computer up-to-date!

- avoid erasing the wrong file
- back-up files by date and time
- keep dated tax records of computer use
- simplify disk housekeeping chores

OPERATION: DateStamper extends CP/M 2.2 to automatically record the date and time a file is created, read or modified. DateStamper reads the exact time from the real-time clock, if you have one; otherwise, it records the order in which you use files. Disks initialized for datestamping are fully compatible with standard CP/M. **INSTALLATION:** Default (relative-clock) mode is automatic. Configurable for any real-time clock, with pre-assembled code supplied for popular models. Loads automatically at power-on. **UTILITIES:** • Enhanced SuperDirectory • Powerful, all-function DATSWEEP file-management program with date and time tagging • Disk-initializer • Installation and configuration utilities **PERFORMANCE:** Automatic. Efficient. Invisible. Compatible.

Requires CP/M 2.2. Uses less than 1K memory. Real-time clock is optional.

When ordering please specify format

8" SSSD, Kaypro, Osborne Formats **$49**

For other formats (sorry, no 96 TPI) add $5.

Shipping and handling **$3**

California residents add **6%** sales tax

MasterCard and Visa accepted

Specialized versions of this and other software available for the Kaypro.
CP/M is a registered trademark of Digital Research, Inc.

Write or call for further information

Plu*Perfect Systems

BOX 1494 • IDYLLWILD, CA 92349 • 714-659-4432

Circle **no. 69** on reader service card.

Listing Four

```
                                        putc('|', stdout);
                                else
                                        putc(' ', stdout);
                                putc('\b', stdout);
                                cary[myx - 1][myy - 1] = 0;
                        }
                        else
                        {
                                revon();
                                putc(' ', stdout);
                                revoff();
                                putc('\b', stdout);
                                cary[myx - 1][myy - 1] = 1;
                        }
                        break;

                case 'Q':                       /* quit */
                case CTRLQ:
                        c = 'Q';
                        break;

                default:
                        putc(BELL, stdout);
                }
        } while (c != 'Q');

        packit(charnum);
        clrscrn();
}


/*
**      drwbox -- draw the character easel.
*/

VOID    drwbox()
{
unsigned        i, j;

        gotoxy(BOXX, BOXY);
        putc('+', stdout);
        for (i = 0; i < WIDTH; i++)
                putc('-', stdout);
        putc('+', stdout);

        for (i = 1; i <= HEIGHT; i++)
        {
                gotoxy(BOXX, BOXY + i);
                putc('|', stdout);
                for (j = 0; j < WIDTH; j++)
                        putc(' ', stdout);
                putc('|', stdout);

        }

        gotoxy(BOXX, YOFF);
        putc('+', stdout);
        for (i = 0; i < WIDTH; i++)
                putc('-', stdout);
        putc('+', stdout);
}


/*
**      unpack -- unpack the selected character.
*/

VOID    unpack(cnum)
unsigned        cnum;
{
```

```
unsigned        i, j, mypack, loj, hij;
struct  chrary  *myptr;

        for (i = 0; i < WIDTH; i++)
                for (j = 0; j < HEIGHT; j++)
                        cary[i][j] = 0;

        myptr = chrs;
        for (i = 0; i < cnum; i++)
                myptr++;

        if (myptr->attrib & 128)
        {
                loj = 1;
                hij = HEIGHT;
        }
        else
        {
                loj = 0;
                hij = (HEIGHT - 1);
        }

        for (i = 0; i < WIDTH; i++)
        {
                mypack = myptr->ary[i];
                for (j = loj; j < hij; j++)
                {
                        cary[i][j] = mypack & 1;
                        mypack >>= 1;
                }
        }
}


/*
**      packit -- pack the selected character.
*/

VOID    packit(cnum)
unsigned        cnum;
{
unsigned        i, j, mypack, loj, hij, wantlo, wanthi, adjacent;
struct  chrary  *myptr;

/* move pointer to appropriate character */
myptr = chrs;
for (i = 0; i < cnum; i++)
        myptr++;

/* determine if upper or lower 8 pins requested */
wantlo = 0;
for (i = 0; i < WIDTH; i++)
        wantlo |= cary[i][0];

if (wantlo)
{
        loj = 0;
        hij = HEIGHT - 1;
        myptr->attrib = 11;
}
else
{
        loj = 1;
        hij = HEIGHT;
        myptr->attrib = 139;
}

/* pack the character */
for (i = 0; i < WIDTH; i++)
{
        mypack = 0;
        for (j = loj; j < hij; j++)
        {
                mypack >>= 1;
                if (cary[i][j])
                        mypack |= 128;
        }
```

(Continued on next page)

```
                myptr->ary[i] = mypack;
        }

        /* determine if upper 8 pins requested */
        wanthi = 0;
        for (i = 0; i < WIDTH; i++)
                wanthi |= cary[i][(HEIGHT - 1)];

        /* determine if adjacent pins requested */
        adjacent = 0;
        for (i = 0; i < (WIDTH - 1); i++)
        {
                if (adjacent = (myptr->ary[i] & myptr->ary[i + 1]))
                        break;
        }

        /* check for definition violation */
        if (adjacent || (wanthi && wantlow))
        {
                gotoxy(0, PROMPTLINE);
                fputs("This definition is illegal.\n", stderr);
                pause();
        }
}

/*
**      flashit -- draw selected character.
*/

VOID    flashit(cnum)
unsigned        cnum;
{
unsigned        i, j;

        for (i = 0; i < WIDTH; i++)
                for (j = 0; j < HEIGHT; j++)
                {
                        boxgotoxy(i + 1, j + 1);
                        if (cary[i][j])
                        {
                                revon();
                                putc(' ', stdout);
                                revoff();
                        }
                        else if (i%2)
                                putc('|', stdout);
                        else
                                putc(' ', stdout);
                }
}


/*
**      boxgotoxy -- goto x,y position in character editing box.
*/

VOID    boxgotoxy(x, y)
unsigned        x, y;
{
        gotoxy(x + BOXX, YOFF - y);
}
```

End Listing Four

(Listing five begins on page 56)

INSIGHT™
EXPERT SYSTEMS

"INSIGHT is essentially the equivalent or better than any other tool available for the personal computer."

Paul Harmon, author of *Expert Systems, Artificial Intelligence in Business*

Turn your PC into an expert.

Give it Insight, or give it Insight 2. Both let you create knowledge base systems using any PC-compatible text editor.

Insight not only simplifies access to lots of information, it analyzes and offers solutions. For entry-level operators it's a perfect procedural training package to help build and implement knowledge base software.

Insight 2 is more than just an "expert." It's a knowledge base engineering tool with application capabilities. It can call up Pascal programs, read and write dBASE II® files, and its decision-making process can tie in directly to your existing databases. Run-only versions also can be developed and distributed.

Two unique packages from the same expert idea.

Level Five Research, Inc.

Insight™ ($95) and Insight 2™ ($485) run on the IBM®PC, DEC® Rainbow, and Victor® 9000.

4980 South A-1-A Melbourne Beach, Florida 32951 (305) 729-9046

Circle **no. 53** on reader service card.

"Gerez vos fenêtres!"*

Introducing MATIS™, the powerful new developmental system from France.

A complete and meticulously detailed program to make a programmer's work easier, faster, and . . . but of course . . . *better.*

☐ Window Management Systems ☐ Screen Generator ☐ Expanded Basic Commands ☐ Can be accessed from other languages ☐ 100% Assembler ☐ Automatic Scrolling in Windows ☐ Virtual Page larger than screen (up to 65534 rows x 65534 columns) ☐ Save or Print Pages ☐ MS-DOS ☐ 170 Page Manual (In *English* Mon Ami!) ☐ Only $150.

ORDER BY MAIL—WRITE OR CALL FOR COMPLETE DESCRIPTION
No license fee.

Softway, Inc.

500 Sutter Street • Suite 222– D • San Francisco, CA 94102
Tel: (415) 397-4666 Telex: 880857

"Manage your windows!"

Circle **no. 92** on reader service card.

RUN/C:™ The *Affordable* C Interpreter

Available NOW for only **$149.95!**

Finally, a painless introduction to the C language. With **RUN/C: The C Interpreter** you can create and run C language programs in an environment as easy to use as BASIC.

RUN/C is C for the rest of us. It is a robust implementation of standard K&R. **RUN/C** is for both the beginner and professional.

RUN/C includes full floating point, 8087 support, structures, unions, casts and more than 100 built-in C functions.

With **RUN/C** you get all this with a command structure modeled after BASIC's using familiar terms such as EDIT, RUN, LIST, LOAD, SAVE, TRON, SYSTEM, etc.

Since **RUN/C** is a true interpreter it means that C programs can be written, tested and run within a single protected environment. It is a teaching tool and a source code debugger.

Here's more good news . . .

- Great documentation: a 400-page, easy-to-read manual filled with executable programs
- Array-index and pointer bounds checking
- Variable-trace and dump diagnostics PLUS an integral program profiler
- Full buffered and unbuffered file I/O
- Printer and asynch support
- Forking to your favorite full screen editor with automatic return to **RUN/C** with your edited program
- System Requirements: IBM® PC or compatible with PC-DOS 2.0 or MS™-DOS 2.0 or greater with ANSI.SYS.

Get things right the first time with **RUN/C: The C Interpreter.™**

For immediate delivery or more information, call:

1-800-847-7078
(in N.Y. **1-212-860-0300**)

or write: Lifeboat Associates™
1651 Third Avenue
New York, NY 10128

Circle **no. 74** on reader service card.

Listing Five

```
/*
**      prnout.c -- the function to print out the entire character
**  set of the FX80 printer.  The function printall() will print the
**  characters taking into account special handling required to
**  print characters stored in the locations of control codes.  The
**  only character not printed is that stored in 127, the alternate
**  zero.  I can't figure out how to do it.
*/


#include        <stdio.h>
#include        <fx80.h>


/*
**      printall -- print the entire character set.
*/
VOID    printall()
{
unsigned        i;

        for (i = 0; i < NUMCHARS; i++)
        {
                if (!(i%64))            /* line feed every 64 characters */
                        pputc('\n');

                if (isprint(i & 0x7f))  /* normal Roman or Italic */
                        pputc(i);
                else if (i >= 128)      /* high order control code */
                {
                        pputc(ESC);
                        pputc('6');
                        pputc(i);
                        pputc(ESC);
                        pputc('7');
                }
                else if (isspecial(i))  /* special low order control code */
                        pspecial(i);
                else                    /* low order control code */
                {
                        if (i == 127)
                                continue;       /* swallow it */

                        pputc(ESC);
                        pputc('I');
                        pputc('1');
                        pputc(i);
                        pputc(ESC);
                        pputc('I');
                        pputc('0');
                }
        }
        pputc('\n');
        pputc('\n');
}


/*
**      isspecial -- return TRUE if argument is low order control
**  code requiring special handling to print.

*/

int     isspecial(i)
unsigned        i;
{
        return ((i >= 7 && i <= 15) || (i >= 17 && i <= 20) ||
                i == 24 || i == 27);
}
```

```
/*
**      pspecial -- print control codes that require special handling.
*/

VOID    pspecial(i)
unsigned        i;
{
        pputc(ESC);     /* all select an international character set */
        pputc('R');

        switch (i)
        {
                case 7:
                        pputc(7);
                        pputc(91);
                        break;
                case 8:
                        pputc(7);
                        pputc(93);
                        break;
                case 9:
                        pputc(7);
                        pputc(92);
                        break;
                case 10:
                        pputc(7);
                        pputc(124);
                        break;
                case 11:
                        pputc(5);
                        pputc(36);
                        break;
                case 12:
                        pputc(7);
                        pputc(35);
                        break;
                case 13:
                        pputc(4);
                        pputc(93);
                        break;
                case 14:
                        pputc(4);
                        pputc(125);
                        break;
                case 15:
                        pputc(1);
                        pputc(92);
                        break;
                case 17:
                        pputc(2);
                        pputc(126);
                        break;
                case 18:
                        pputc(4);
                        pputc(91);
                        break;
                case 19:
                        pputc(4);
                        pputc(123);
                        break;
                case 20:
                        pputc(4);
                        pputc(92);
                        break;
                case 24:
                        pputc(2);
                        pputc(92);
                        break;
                case 27:
                        pputc(5);
                        pputc(124);
        }

        pputc(ESC);             /* Return to USA font */
        pputc('R');
        pputc(0);
}
```

End Listings

A Magic Mushroom for the Epson Alice

A Practical Program for Printing Phonetic Characters

by Gary B. Palmer

I am using an Apple II Plus microcomputer and an Epson MX-100 printer with Graftrax Plus to print stories of the Coeur d'Alene Indians in their native language; this requires phonetic characters. Still spoken fluently by a handful of elderly Indians on the reservation in northern Idaho, Coeur d'Alene belongs to a family of languages known for their difficult phonetic systems. Linguists who record Indian languages of the American Northwest have designed a character set that is visually elegant but extremely difficult to reproduce on a typewriter.

Table 1 (page 59) shows the Coeur d'Alene characters. Entering a single phonetic character with a customized linguistic element may require the typist to use as many as six keystrokes, often involving keys located in awkward positions. With phonetically complex languages, the tedium and frustration of keyboard entry send the probability of errors soaring.

An exploration of phonetic printing and rubberized fonts for the Epson MX-100.

To those of us working with Indian languages, microcomputers promise convenient transcription, rapid processing, and accurate printing.

I begin on the reservation by recording the stories on audio cassette tape. After transcribing a text, using phonetic symbols, I work with the Coeur d'Alenes to translate the story into English. I use the word processor Applewriter II to enter both the Indian text and the translation into the computer, but I do not use the computer to translate the text.

What a Linguist Needs

Automatic printing of bilingual texts on the Epson MX series of printers requires graphics software. Such software is readily available for the Apple Macintosh, which prints excellent characters on the ImageWriter dot matrix printer. Other late model dot matrix printers, such as the Epson FX series that replaced the MX, permit loading of user-defined fonts. This is a great convenience for linguists, but if your available printer belongs to the MX series, that is small comfort. Although the MX-100 does not permit loading of fonts, you can still print user-defined fonts by sending graphics commands to the printer.

Linguists have a great interest in displaying alternate character sets on the CRT screen. On the Apple II, this is easily accomplished with the appropriate 80 column card or with a patch to the word processor's character set if the characters are created by the software. But when you wish to produce a manuscript, the creation of alternate character sets on the printer becomes an issue. It is important to realize that the printer and the screen are independent, a fact that is not at all transparent to newcomers.

Linguists who wish to switch from one language to another at will, mix-

Gary B. Palmer, Center for Computer Applications in the Humanities, University of Nevada, Las Vegas, NV 89154.

ing codes in lines of text, will discover that the limitations of the standard keyboard require them to select a set of characters to represent the phonetic symbols for text entry: each key must correspond to an ASCII code number—the only kind of signal acceptable to the printer.

Only 52 keys are available for alphabetical use on the Apple keyboard (ASCII numbers 65 to 90 and 97 to 122). Moreover, the Apple II Plus will send only the first 128, not enough to support an additional concurrent set even if the new set shares many symbols with the standard set. Because the control codes below ASCII 32 are needed to drive the printer, and because standard punctuation marks, numerals, and underlining are commonly needed in the text, the set of characters that can be sacrificed to provide alternative characters is quite small: #, $, %, &, *, +, /, <, =, >, @, [,], ^, {, ¦, }, ~. Of these, several (@, [,], ^, {, ¦, }, ~,) require awkward combinations of shift and control characters that are difficult to remember and to enter, at least with Applewriter II Plus. This inconvenience defeats our original goal of facilitating manuscript preparation.

How To Get It

The solution lies not on the keyboard but in the software. In the text file, the standard character set can represent the phonetic set; a printer driver written in APPLESOFT BASIC defines the phonetic characters. Code inserted in the text signals character set changes. A signal to change sets can be entered as one or more nonalphanumeric symbols, and each unique signal makes available at least 52 new shapes. Because the printer can create a practically unlimited number of shapes, it is possible to access many alternate character sets from a single text.

A special problem arises if a character exceeds the standard 7/72-inch height. Because the MX series lacks reverse line feeds and expanded character matrices, the vertical extension must be printed before the remainder of the character. This means the upper story of *every* extended character on a line must be printed before advancing the line feed.

The program described in this article creates vertically extended fonts and generates one alternate character set that can be mixed at will with English. The program does word wrap, avoids breaking words at the end of lines, numbers pages, and permits variable line spacing. Definitions are entered as data lines (twelve data points/character).

To produce a phonetic symbol, it gets one character at a time from the text file, replacing the ASCII number with a string of graphics commands. Each string of code, representing a phonetic symbol, is placed in one element of an array of 60, the length of a line. Decoding and printing the graphics set is slower than printing English but still very practical compared to performing the same task with a modified typewriter.

Convenient Text Entry

Entry codes must be easy to remember and to type. I use a system that assigns a single key to frequently occurring phonetic features, such as stressing and rounding. This avoids the necessity of remembering a separate key for each stressed vowel or rounded consonant.

The system requires no unusual substitutions (for one who understands Coeur d'Alene phonetics); it does not resort to otherwise little-used capital Zs , Qs, numbers, or special keys. These can all be kept for their normal uses or for character set change flags. Table 2 (below) lists some representative characters from

These are the Coeur d'Alene vowels: a ɛ ə i ɪ o ɔ u.
These are the Coeur d'Alene consonants: b c c̓ č č̓ d gʷ h ǰ kʷ k̓ʷ l l̓ ɬ m m̓ n n̓ p p̓ q q̓ qʷ q̓ʷ r r̓ R R̓ Rʷ R̓ʷ s š t t̓ w w̓ xʷ x̱ x̱ʷ y y̓ ʔ. These are English vowels often found in Coeur d'Alene texts: a ɛ ə i ɪ ɔ o u ʌ. These are English consonants often found in Coeur d'Alene texts: ð θ ŋ. Any vowel can be printed with stress, as in the following: á ɛ́ ə́ í ɪ́ ó ɔ́ ú ʌ́.

Table 1
Coeur d'Alene Phonetics

CONSONANTS Entry	WPL	Phonetic	VOWELS Entry	WPL	Phonetic
c	c	c	e	e	ɛ
c'	c'	c̓	e!	e!	ɛ́
C	C	č	E	E	ə
C'	T	č̓	i!	i!	í
gw	gW	gʷ	I	I	ɪ
'	H	ʔ	o	o	ɔ
j	j	ǰ	O	O	o
kw	kW	kʷ	u	u	u
l'	l'	l̓			
L	L	ɬ			
q'	q'	q̓			
qw	qW	qʷ			
q'w	q'W	q̓ʷ			
R	R	R			
R'	R'	R̓			
Rw	RW	Rʷ			
R'w	R'W	R̓ʷ			
s	s	s			
S	S	š			
w	w	w			
xw	xW	xʷ			
X	X	x̱			
Xw	XW	x̱ʷ			

Table 2
Phonetic Characters for the Coeur d'Alene Language, in Entry Code, after Processing by A WPL Filter Program, and in Final Form.

the set of Coeur d'Alene phonetics.

All entries bear some resemblance to the actual phonetic symbols that they represent. All are either lower-case letters, capitals, or simple double keystrokes called "digraphs." Speakers of Coeur d'Alene pronounce several phonemes with a catch in the throat, so I appended the apostrophe symbol to those phonemes for all such glottal sounds. Likewise, several phonemes are produced with a rounding of the lips, so I appended the lower-case w for all such labial sounds. Because the symbols are easy to remember and the same keyboard pattern serves for both languages, text entry is simple.

Any good word processor permits similar entry, but Applewriter II has another useful feature: it comes with its own programming language, called Word Processing Language (WPL), which is designed for string manipulation. When preparing long text files for the printer driver, I use a WPL filter to convert some digraphs into single characters. It also prefixes lines of English and Coeur d'Alene with change codes. Table 2 shows the two transformations required to produce phonetic text from my entry code. Although not strictly necessary, the WPL filter does simplify text entry.

Graphics Fonts on the MX-100

A few hours of experimentation will disclose the difficulty of printing text files with Epson graphics characters. Taking pity on me, a colleague called my attention to the Gutenberg software, which he thought might work. Having just stretched the department budget to purchase a computer and printer, however, we could not go back again for an unproven software item costing over $300. For those with a CP/M card, the Fancy Font program would serve the purpose, but the cost made this option unavailable to us. Not wishing to risk buying software of uncertain utility, I opted for writing a printer driver: a program to act as a filter to replace ASCII characters with graphics.

Creating special fonts requires some basic arithmetic. On the MX-100, English consonants normally occupy up to 7/72 inch above the baseline and 2/72 inch below (see Figure 1, below). The remaining 3/72 inch in each normal line feed of 12/72 inch provides the space between lines. All vowels extend only 5/72 inch above the baseline except i, which extends 7/72 inch. Each character is printed within the space of five columns of dots. A trailing sixth column provides spacing between letters.

Standard fonts are mapped in the printer, and when the printer slot is addressed (normally with PR#1), a single ASCII code number prints a character. For example, the command PRINT CHR$(65) prints A. The Apple printer and newer Epson printers such as the FX-80 can load user-defined fonts and then treat them in much the same way as standard character sets. With the MX-100, however, alternate fonts require graphics code from the computer each time a character is printed.

On the MX-100, each column in a

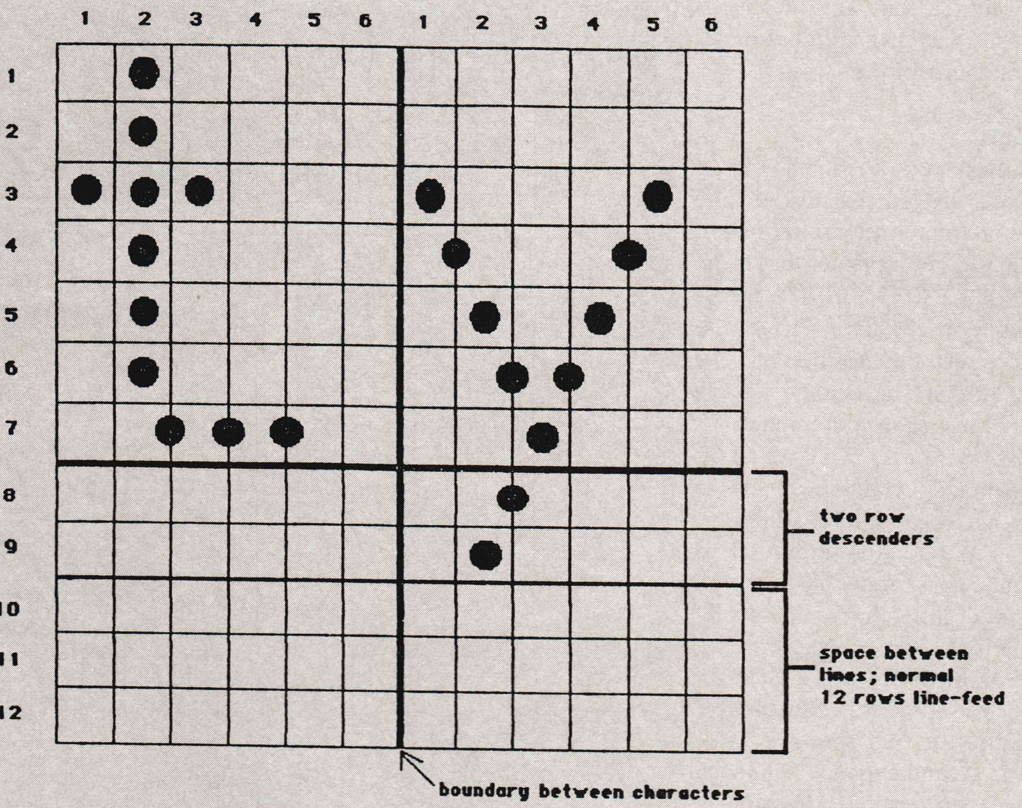


Figure 1
Standard Lower-Case t and y on the Epson MX-100

character's six-column matrix is sent with a code number. Code specifying the shape of these columns must be prefixed by a graphics command. A high-density mode also must be called to generate well-defined characters; because this mode moves the printing head only half a column at a time, the graphics command creates twelve-column characters. Take as an example the command: PRINT CHR$(27) + CHR$(76) + CHR$(12) + CHR$(0). CHR$(27) says to treat the following characters as commands rather than ASCII symbols. CHR$(76) calls the high-density graphics mode. CHR$(12) sets aside twelve columns of dots. CHR$(0) signifies that less than 256 columns follow.

The high-density mode produces definition and density equal to that of standard fonts printed in emphasized mode. In the printer driver (see Listing One, page 66), the twelve data points necessary to define the overlapping columns of a phonetic symbol are provided in DATA lines 1050 through 1170. Each data point represents a column seven dots high. Figure 2 (page 63) illustrates data points for the graphics character "schwa." Although the Epson MX-100 manual describes graphics procedures, it does not mention the use of the high-density mode for designing attractive fonts.

The complete character is constructed by concatenation. As the READ G statement in line 1000 finds each data point, the statement G$(M) + CHR$(G) appends it to the string of data that represents the twelve printed columns. Then the twelve columns in G$(M) are appended to the graphics command GC$ in the statement G$(M) = GC$ + G$(M). Finally, the command PRINT G$(M) will print the phonetic symbol. However, this command is never given explicitly, because the value of G$(M) is first transferred to yet another array element that represents the top or bottom sector of a character.

Although standard characters can extend as high as 7 rows above the baseline, a graphics column can be only 6/72 of an inch high. Likewise, the graphics column extends only 1/72 of an inch below the baseline, compared to the 2/72 drop of the standard j, g, or y. This is because the Apple II Plus can address only seven bits; it prints only six dots above and one dot below. For many purposes, seven rows will suffice, but if one needs taller fonts, there is a solution.

Vertical Extension of Fonts

Some applications, such as Chinese characters, are too complex for a seven-by-six dot matrix. But for Epson Alices, the printer driver offers a magic mushroom: smaller line feeds can make fonts grow infinitely tall. Two normal lines spliced together make one tall line, and ten lines spliced together make a very tall line. Split characters are aligned horizontally by printing lines in multiple arrays of corresponding elements and are joined vertically by controlling line feeds.

In the Coeur d'Alene set, diacriticals signify phonetic features; the most frequent is the stress mark on vowels, represented in entry code with !. If the MX-100 supported reverse line feeds, we could backspace with PRINT CHR$(8);, send a one-half reverse line feed, print the character, and send a half line feed for-

ward. Because reverse line feeds are unavailable, the stress mark must be printed first, followed by a partial line feed, and then the vowel.

Similarly, each character with a vertical extension has a top sector and a bottom sector. If the stress mark on top is T$(1), the vowel on the bottom is B$(1), and PLF$ represents a partial line feed command, then PRINT T$(1):PRINT PLF$:PRINT B$(1) will print the full phonetic symbol.

I designed the diacriticals so that a partial line feed of 4/72 inch adjusts them to vowels of standard height (see Figure 3, page 63). The command is TF$ = CHR$(27) + CHR$(65) + CHR$(4) (see line 1240 in the listing). Following the escape character, CHR$(65) resets the line spacing and CHR$(4) sets the spacing at four rows of dots.

To avoid a cascade effect, the program prints an entire line of diacriticals first, followed by a partial line feed, and then a line of base characters. The diacriticals must be stored in the array T$(X), where X is the character counter. Baseline characters are stored in the array B$(X). If a phonetic symbol lacks a diacritical, a blank must be printed in the top sector to maintain proper alignment between the top and bottom of subsequent complex symbols; this is accomplished by initializing the whole T$(X) array with blanks before getting a line.

The code for a stress mark, !, is entered into the text file after its vowel. We then send the corresponding graphics mark to its proper destination over the vowel by placing it in T$(X – 1) rather than T$(X). X is decremented by one so the next character will be assigned to T$(X) rather than T$(X + 1) (see line 200 in the listing). The same procedure is used for the graphics apostrophe in line 250.

When vertically extended characters are represented by a single character in the text file, both the top and bottom sectors of the character can be assigned directly to corresponding array elements. For example, T represents the standard c with a diacritical cap that looks like a v with a nested apostrophe. The graphics cap is assigned to the T$(X) array in line 270, while the standard Epson c goes into B$(X) (see Figure 4, page 63). A column can be raised one row by doubling its number. To raise an entire character one row, double the data point values for each of the twelve columns.

The example in Figure 4 illustrates vertical extension by setting a graphics character on top of a standard font. It would work just as well in reverse, if that produced the desired shape, or you could use graphics in both the top and the bottom sectors, as I have done with the character "eth" (see Figure 5, page 63). Overstriking, superscripting, and subscripting provide additional possibilities accessible to either the top or bottom sectors in this flexible system; however, I have only been able to achieve overstriking on standard characters.

The Program

Because characters consume more memory than numerals, text processing runs more slowly than calculations. You can accelerate a program by placing frequently used subroutines near the beginning and by storing string constants in variables; initialization routines are placed near the end. Thus, program execution first GOSUBs to line 850, where it sets the margins, length of lines, and flags, then it dimensions the top and bottom sector arrays, T$(X) and B$(X).

Beginning with line 980, two nested FOR/NEXT loops define the phonetic set. The inside or N loop beginning on line 990 reads one data line to build a graphics string by concatenation; each data point represents one high-density column. At line 1030, the column descriptor string is prefixed with the graphics command. The outside or M loop beginning with line 980 cycles the process through all the data lines.

Commonly used sequences of printer commands for emphasis, superscripting, and partial and normal line feeds are assigned to string variables from line 1190. This segment also defines the two overstrike characters ɫ and x.

From line 1320, a DOS subroutine catalogs the text diskette, prompts for the name of the text file, verifies the name, opens the file, and opens the printer port, PR#1. Notice in lines 1500 and 1510 that the command to the printer precedes the PRINT D$;"READ";T$ command to the disk drive. Why this particular order is necessary remains one of those mysteries of the silicon cosmos, but the program will not work properly if the statements are reversed.

After the catalog has been listed and the file name entered, the program prompts for the desired line spacing with a default value of 0 for single spacing. The IF/THEN statement at line 1500 rejects spacings less than zero or greater than nine.

Before getting the first character from the text file, the program performs one more task: a subroutine beginning on line 650 assigns blanks to all elements of the T$(X) and B$(X) arrays. The blanks in the top sector remain, unless replaced by diacriticals or partial fonts, to guarantee proper horizontal alignment of diacriticals to character base sectors.

After initialization, execution drops to line 100 to get the first character (A$) from the text file and to increment the character counter X. Because both arrays have been initialized with blanks, a blank in A$ does not require an assignment; execution is routed to line 90, which checks for line length. If A$ is not a blank, lines 120 and 130 check whether A$ is a flag for English (ENG$ = "^") or for Coeur d'Alene (CDA$ = "+"). If A$ is a flag, the result is stored in FLAG$ and execution returns to line 100 to get another character.

Because flags are not printed, the counter is decremented by one. The flag variable in line 150 directs subsequent traffic. When FLAG$ is English, A$ is assigned directly to an element of the B$(X) array at line 150; T$(X) remains a blank, as initialized. When the flag variable switches to Coeur d'Alene, execution branches to line 180 where a graphics or backspace-and-overstrike sequence corresponding to A$ is assigned to B$(X).

All paths to the printing subroutine pass through carriage-return and end-of-line checks. If A$ is a carriage

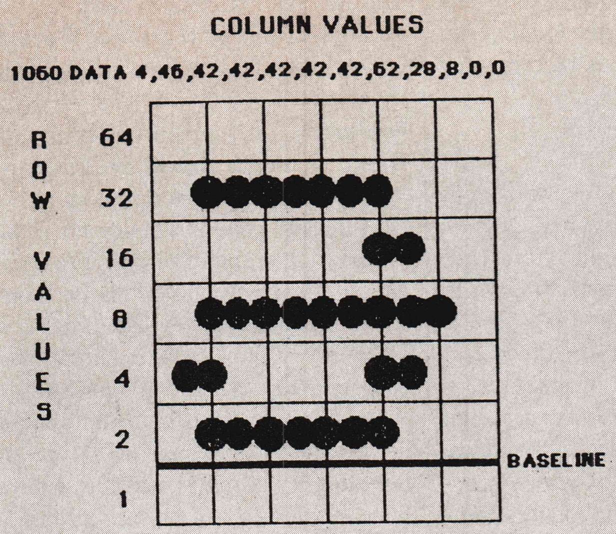


Figure 2
Data Points for the Character "Schwa" (Compare column values with Listing One, line 1060.)

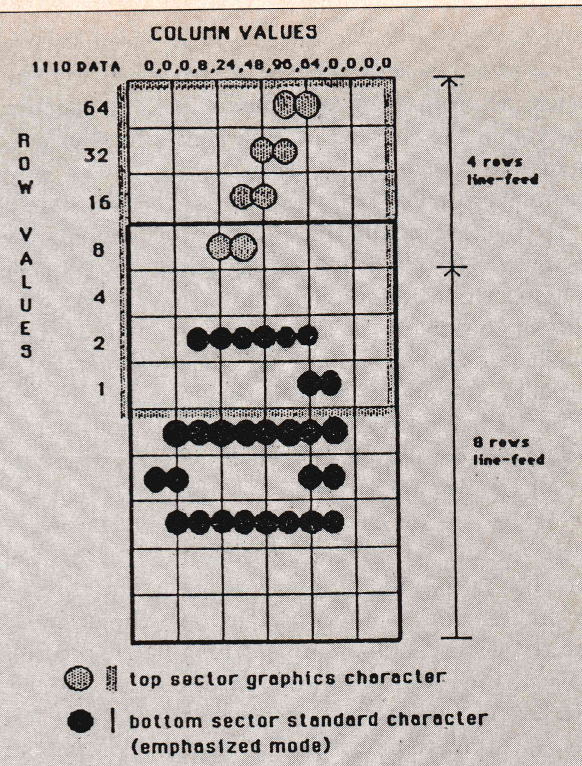


Figure 3
Alignment of Graphics Stress Character by Means of Partial Line Feed (Compare Listing One, lines 200 and 1110.)

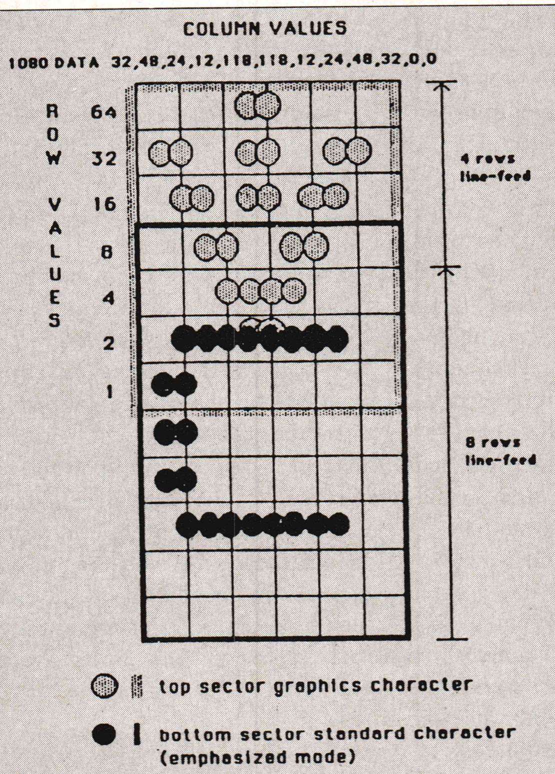


Figure 4
Partial Overlay of Graphics Diacritical on Standard Lower-Case c to Print Composite Character (Compare Listing One, lines 270 and 1080.)

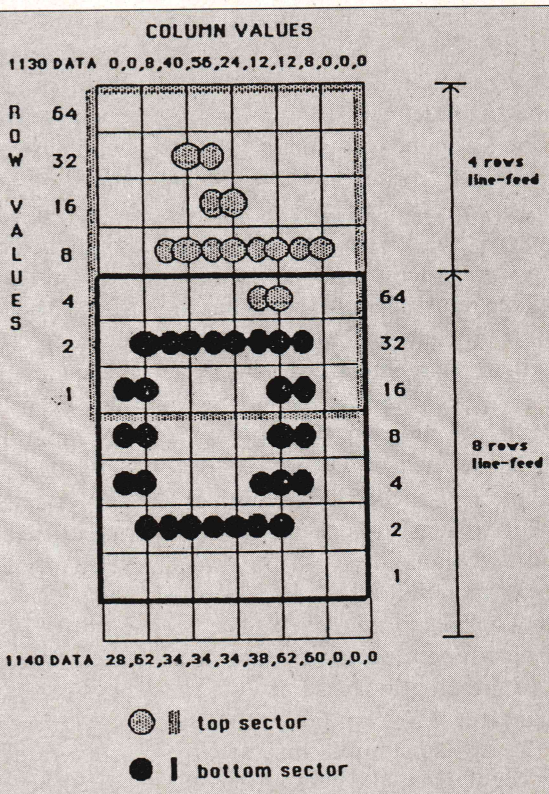


Figure 5
Vertically Extended Graphics Character "Eth" (Compare Listing One, lines 320, 1130, 1140.)

return, the end-of-line variable EOL is set as the current value of X. Because no word segment remains at the end of the line, the variable SEG is set to 0. Execution branches to the printing routine at line 410.

If a line contains more than 60 characters, the check at line 90 causes a branch to line 690; there a reverse checking procedure looks for the last blank in the line to set the end-of-line variable. It also determines the length of the trailing word segment to be carried over to the next line.

When execution passes to the printing subroutine at line 410, PRINT SPC(LM); prints the left margin. The FOR/NEXT loop beginning in program line 450 prints the top sector diacriticals contained in T$(X). Line 460 prints the partial line feed of four rows and increments the line feed counter, LF. To complete a line, it remains only to print the baseline characters. PRINT BF$; resets the line spacing to eight rows. For unknown reasons, the eight-row line feed cannot be printed with a single print command as is done for the four-row line feed. The left margin is printed again, and the FOR/NEXT loop beginning at program line 500 prints the baseline.

The assignments made in the loop at line 610 transfer the line-final word segment to the beginning of the following line of text. The loop beginning with program line 660 reinitializes the remainder of the two line arrays. Subsequent lines set X to the length of the segment and reinitialize the counters FIRST and SEG.

Now the program cycles back to the line feed check in line 80. More arithmetic is needed. The length of a page is 66 lines; after the last line, we print two blank lines, a line number, and six more lines for a total of nine with a bottom margin of six. The figure nine represents the bottom margin in the program. At line 780, the assignment FILL = 66 − BM − LINES says to subtract nine and the line count from 66 lines. After the command PRINT NLF$ returns the line feed status to normal, the fill lines and bottom margin are printed. Line 800 centers the page number, which is printed in line 810. The line feed counter is reinitialized to the value of the top margin (four), and the page counter PG is incremented by one.

Summary

The program in the listing prints vertically extended phonetic characters on the Epson MX-100 with Graftrax Plus by storing graphics commands in two arrays for each line. Flags inserted in the text file route ASCII codes to English or phonetic program segments. In the phonetic branch, the ASCII codes are replaced by graphics, overstrike, or superscript code sequences. A phonetic symbol may combine these modes with standard characters.

Formatting features include margins, page numbering, word wrap, and variable line spacing. English and phonetic fonts mix freely within lines of text. While perhaps not as flexible as graphics font programs that print one row at a time, a program such as this, which overlays whole lines, should print more rapidly—an important consideration when long texts are involved. If you use the same character set repeatedly, the once-only effort of programming data lines is minor compared to the advantages of using a familiar word processor for text entry and formatting.

Although these graphics routines might be unnecessary with new printers that support downloading of fonts, the system of storing partial characters in arrays and splicing lines by means of partial overlays is adaptable even to these printers. Because the program stores a phonetic character as a string of commands, it is also adaptable to daisy wheel printers, especially those that support reverse line feeds. A Coeur d'Alene orthography similar to the one created on the Epson could be printed on a daisy wheel printer with only three physically modified characters. The program would require only one line array, rather than the two arrays required to print vertically extended symbols on the MX-100.

Although many Chinese or Kanji characters would fit within the vertical format of two arrays, large and complex fonts of Chinese, Japanese, or other symbol systems may require splicing of multiple arrays. You can create wider characters by using more than twelve columns in the graphics command prefix. For example, a Kanji character could probably fit within a matrix of 16 dot rows by 32 high-density columns. Aside from the problem of graphics design, practical use of fonts with hundreds or thousands of characters would require a different system of text entry and character-to-keyboard assignment.

Once you understand the binary method of specifying the shape of high-density graphics columns in data lines, adaptation of this program to print other two-sectored fonts is relatively simple. Owners of MX series Epson printers with an interest in special character sets may find it essential. Modification of escape sequences will make it available to other printers. If you have a secret urge to print Sanskrit, runic, or Elf script, the tools are now available. Please send me listings of your data statements.

Postscript

Listing Two (page 70) contains some of the more commonly used special characters of the International Phonetic Alphabet taken from Robert Albright's article "The International Phonetic Alphabet" (*International Journal of American Linguistics,* vol. 24, no. 1, January 1958). The program and data lines can be substituted as needed into the sections of Listing One titled "*Assign Fonts To Array Elements*" and "*Graphics Fonts.*" The choice of an appropriate entry character for each graphics character will depend upon the organization of the user's orthography. Few applications would require all the characters. Avoid the numbers 7, 8, and 9 in graphics data statements for the Epson MX printers; they result in beeps and unpredictable graphics.

DDJ

(Listing begins on page 66)

THE fifth generation language

P R O L O G

Implementing the full Edinburgh Syntax as described by Clockson and Mellish. Recognized by Japan as providing unparalleled opportunity for artificial intelligence.

Applications:

- ▲The highest level of a heirarchial robotic control system.
- ▲Machine recognition of natural language.
- ▲Expert systems and knowledge engineering.

Optional:

- ▲Special libraries
- ▲Language extensions

Educational Package

$29.95

(MSDOS)

Highlights of LVM Prolog:

- ▲Type LVM Prolog makes possible the execution of AI applications previously only possible on a mainframe.
- ▲Invisible compilation to a semantic network provides the flexibility of the interpreted mode and the speed of a compiler.
- ▲Virtual memory and a sophisticated cache algorithim limited only by the DOS, typically 2 gigabytes.
- ▲Syntax superset with many extensions such as pattern specified insertion and deletion, clause indexing, robotic control.
- ▲Unlimited number of resident and virtual modules, each up to 2 gigabytes in size.

▲THE most cost efficient package in the industry.

$500- (MSDOS version) Also available for Xenix, Unix, CP/M68K

automata design assoc.

▲VISA, Mastercard, AMEX

▲call or write for brochure

1570 Arran Way Dresher, PA 19025 technical: (215)646-4894 orders: (215)355-5400

Circle **no. 130** on reader service card.

ConIX™

UNIX™ Technology for CP/M™

ConIX can provide any 48K+ CP/M-80 compatible system with many advanced capabilities of UNIX. You'll be amazed at what your 8-bit micro can do now! ConIX features include:

I/O Redirection and Pipes (uses memory or disk), multiple commands per line, full upper/lower case and argument processing, Auto Screen Paging, Programmable Function Keys, improved User Area Directory manipulation, Command and Extension (Overlay) Path Searching, "Virtual" disk system, 8Mb Print Spooler, extensive preprocessed "Shell" command programming language, 300+ variables, over 100 built-in commands, Math Package, 22 new BDOS SysCalls, Archiver (compacts files for disk space savings of over 50%), On-Line Manual System, and much more! Uses as little as 1/2K RAM! Runs with CP/M for true data and software compatibility. Installs quickly and easily without any system modifications.

The ConIX Operating System
List Price: $165

Price includes Instructional Manual, 8" SSSD disk, and free support. 5¼" format conversions available. To order, contact your local dealer, or buy direct and add shipping: $4.50 UPS, $10 Canada, $25 overseas, COD $2 extra (USA only). NY State residents add sales tax.

Computer Helper Industries Inc.
P.O. Box 680 Parkchester Station, NY 10462
Tel. (212) 652-1786

Dealer inquiries invited!

UNIX: AT&T Bell Labs, CP/M: Digital Research, ConIX: Computer Helper Ind.

Circle **no. 22** on reader service card.

Checks & Balances™

is creating Panda-monium

for Home Accountant and Quick Check!

When people compare Checks & Balances to **Home Accountant** and **Quick Check,** it's no contest! Just look at these examples:

•**We** give you powerful, easy to use and easy to remember English commands to take you from one function to another instantly. **They** give you a menu to go to another menu which goes to another menu to go to another menu. . . .

•**We** put up to seven entries on a screen at a time in a form just like your check register and let you scroll forward and back like flipping pages. **They** disply one entry per screen that doesn't look at all like your familiar check register.

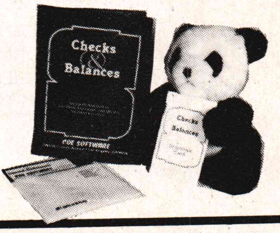
•**We** let you make corrections on any entry displayed by inserting or deleting characters or typing over the error just like your word processor. With **them** making corrections means going to special menus where you have to call up the entry again, then going back through the menus to resume your work.

•**We** give you over forty characters of space for the payee and memo instead of as few as fifteen.

•**And best of all . . . we** cost only **$74.95. They** cost a whole lot more!

CHECKS & BALANCES also has a Rolodex for names, addresses, phone numbers and notes, can print nearly any type of check, has reports comparing income and expenses, lets you disperse any entry to multiple categories, keeps a full year at a time, and gives you as many checkbooks as you want, on any disk drive.

Start the New Year out right (and get last year in order for tax time). Pick up **CHECKS & BALANCES** at your local dealer, or order direct from CDE. **CHECKS & BALANCES** is available for PC-DOS, MS-DOS and CP/M. Indicate computer type, operating system, and disk type (8" SSSD, 5¼"SSDD, or 5¼" DSDD). California residents include 6% sales tax, and everyone include $2 shipping and handling. Dealer distribution bundling rates available. Write for complete catalogue of software for home and business.



Visa/MC accepted **CDE SOFTWARE** (213)-661-2031

2463 McCready Ave. • Los Angeles, CA 90039

Circle **no. 13** on reader service card.

Magic Mushroom for the Alice (Text begins on page 58)

Listing One

```
10  ONERR GOTO 1630
20  TEXT : HOME
30  REM ---------------------------------------------------------------
40  GOSUB 850: GOSUB 650: REM INITIALIZE
50  REM ---------------------------------------------------------------
60  REM              GET CHARACTER AND SET LANGUAGE FLAGS
70  REM ---------------------------------------------------------------
80  IF LINES >  = 66 - BM THEN  GOSUB 760: REM END-OF-PAGE
90  IF X = LL THEN  GOSUB 690: GOSUB 410: GOTO 80: REM EOL,PRINT,PG
100 GET A$: X = X + 1
110 IF A$ = " " THEN 90
120 IF A$ = ENG$ THEN FLAG$ = ENG$: X = X - 1: GOTO 100
130 IF A$ = CDA$ THEN FLAG$ = CDA$: X = X - 1: GOTO 100
140 IF A$ = CR$ THEN EOL = X:SEG = 0: GOSUB 410: GOTO 80: REM PRINT
150 IF FLAG$ = ENG$ THEN B$(X) = A$: GOTO 90
160 GOSUB 180: GOTO 90
170 REM ---------------------------------------------------------------
180 REM                 ASSIGN FONTS TO ARRAY ELEMENTS
190 REM ---------------------------------------------------------------
200 IF A$ = "!" THEN :T$(X - 1) = G$(7):X = X - 1: RETURN
210 IF A$ = "E" THEN B$(X) = G$(2): RETURN
220 IF A$ =  CHR$ (101) THEN B$(X) = G$(13): RETURN
230 IF A$ = "W" THEN B$(X) = SS$: RETURN
240 IF A$ = "H" THEN B$(X) = G$(1): RETURN
250 IF A$ = "'" THEN T$(X - 1) = G$(6):X = X - 1: RETURN
260 IF A$ = "C" THEN B$(X) =  CHR$ (99):T$(X) = G$(5): RETURN
270 IF A$ = "T" THEN B$(X) =  CHR$ (99):T$(X) = G$(4): RETURN
280 IF A$ = "L" THEN B$(X) = LT$: RETURN
290 IF A$ = "S" THEN B$(X) =  CHR$ (115):T$(X) = G$(5): RETURN
300 IF A$ = "X" THEN B$(X) = BX$: RETURN
310 IF A$ = "I" THEN B$(X) = G$(3): RETURN
320 IF A$ = "D" THEN T$(X) = G$(9):B$(X) = G$(10): RETURN
330 IF A$ = "G" THEN B$(X) = "O" +  CHR$ (8) + "-": RETURN
340 IF A$ =  CHR$ (106) THEN B$(X) = A$:T$(X) = G$(5): RETURN
350 IF A$ = "N" THEN B$(X) = G$(8): RETURN
360 IF A$ =  CHR$ (111) THEN B$(X) = G$(11): RETURN
370 IF A$ = "O" THEN B$(X) =  CHR$ (111): RETURN
380 IF A$ = "U" THEN B$(X) = G$(12): RETURN
390 B$(X) = A$: RETURN
400 REM ---------------------------------------------------------------
410 REM                              PRINT
420 REM ---------------------------------------------------------------
430 REM ** PRINT TOP SECTOR
440 PRINT  SPC( LM);: REM LEFT MARGIN
450 FOR PR = 1 TO EOL - 1: PRINT T$(PR);: NEXT
460 PRINT TF$: REM 4 ROWS LINE-FEED
470 PRINT BF$;: REM 8 ROWS LINE-FEED
480 REM ** PRINT BOTTOM SECTOR
490 PRINT  SPC( LM);: REM LEFT MARGIN
500 FOR PR = 1 TO EOL - 1: PRINT B$(PR);: NEXT
510 PRINT :LINES = LINES + 1
520 REM LINE-SPACING
530 IF S = 0 THEN  GOTO 590
540 PRINT NLF$;
550 FOR L = 1 TO S
560 PRINT :LINES = LINES + 1
570 IF LINES >  = 66 - BM THEN  GOSUB 760: GOTO 590: REM PAGINATION
580 NEXT
590 REM ** CARRY OVER WORD SEGMENT
600 IF SEG = 0 THEN 650
610 FOR FIRST = 1 TO SEG
620     T$(FIRST) = T$(EOL + FIRST)
630     B$(FIRST) = B$(EOL + FIRST)
640 NEXT FIRST
650 REM ** INIT ARRAYS
660 FOR X = FIRST TO LL:T$(X) = " ":B$(X) = " ": NEXT
670 X = SEG:FIRST = 1:SEG = 0: RETURN
680 REM ---------------------------------------------------------------
690 REM                        FIND END-OF-LINE
700 REM ---------------------------------------------------------------
710 FOR EOL = LL TO 1 STEP  - 1
720     IF B$(EOL) = " " THEN SEG = LL - EOL: RETURN
730 NEXT EOL
```

(Continued on page 68)

DSD 80

FULL SCREEN SYMBOLIC DEBUGGER

"THE SINGLE BEST DEBUGGER FOR CP/M-80. A TRULY AMAZING PRODUCT."

LEOR ZOLMAN
AUTHOR OF BDS C

Complete upward compatibility with DDT
Simultaneous instruction, register, stack & memory displays
Software In-Circuit-Emulator provides write protected memory, execute only code and stack protection.
Full Z80 support with Intel or Zilog Mnemonics
Thirty day money back guarantee
On-line help & 50 page user manual

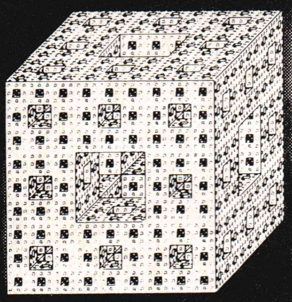
NOW ONLY $125.

SOFTADVANCES

P.O. BOX 49473 AUSTIN, TEXAS 78765 (512) 478-4763

MasterCard VISA

Circle no. 63 on reader service card.



WALTZ LISP (TM)

*The one and only **adult** Lisp system for CP/M users.*

Waltz Lisp is a very powerful and complete implementation of the Lisp programming language. It includes features previously available only in large Lisp systems. In fact, Waltz is substantially compatible with *Franz* (the Lisp running under *Unix*), and is similar to *MacLisp*. Waltz is perfect for Artificial Intelligence programming. It is also most suitable for general applications.

Much faster than other microcomputer Lisps. • Long integers (up to 611 digits). Selectable radix • True dynamic character strings. Full string operations including fast matching/extraction. • Flexibly implemented random file access. • Binary files. • Standard CP/M devices. • Access to disk directories. • Functions of type lambda (expr), nlambda (fexpr), lexpr, macro. • Splicing and non-splicing character macros. • User control over all aspects of the interpreter. • Built-in prettyprinting and formatting facilities. • Complete set of error handling and debugging functions including user programmable processing of undefined function references. • Virtual function definitions. • Optional automatic loading of initialization file. • Powerful CP/M command line parsing. • Fast sorting/merging using user defined comparison predicates. • Full suite of mapping functions, iterators, etc. • Assembly language interface. • Over 250 functions in total. • The best documentation ever produced for a micro Lisp (300+ full size pages, hundreds of illustrative examples).

Waltz Lisp requires CP/M 2.2, Z80 and 48K RAM (more recommended). All common 5'' and 8'' disk formats available.

ProCode (TM)
INTERNATIONAL

15930 SW Colony Pl.
Portland, OR 97224

Unix* Bell Laboratories.
CP/M* Digital Research Corp.

Version 4.4 (Now includes Tiny Prolog written in Waltz Lisp.)

$169*

*Manual only: $30 (refundable with order). All foreign orders: add $5 for surface mail, $20 for airmail. COD add $3. Apple CP/M and hard sector formats add $15.

Call free **1-800-LIP-4000** Dept. #11
In Oregon and outside USA call 1-503-684-3000

Circle no. 73 on reader service card.

LISP

FOR THE IBM PERSONAL COMPUTER.

THE PREMIER LANGUAGE OF ARTIFICIAL INTELLIGENCE FOR YOUR IBM PC.

- **DATA TYPES**
  Lists and Symbols
  Unlimited Precision Integers
  Floating Point Numbers
  Character Strings
  Multidimensional Arrays
  Files
  Machine Language Code
- **MEMORY MANAGEMENT**
  Full Memory Space Supported
  Dynamic Allocation
  Compacting Garbage Collector
- **FUNCTION TYPES**
  EXPR/FEXPR/MACRO
  Machine Language Primitives
  Over 190 Primitive Functions
- **IO SUPPORT**
  Multiple Display Windows
  Cursor Control
  All Function Keys Supported
  Read and Splice Macros
  Disk Files
- **POWERFUL ERROR RECOVERY**
- **8087 SUPPORT**
- **COLOR GRAPHICS**
- **LISP LIBRARY**
  Structured Programming Macros
  Editor and Formatter
  Package Support
  Debugging Functions
  .OBJ File Loader
- **RUNS UNDER PC-DOS 1.1 or 2.0**

IQLISP

5¼" Diskette and Manual	$175.00
Manual Only	$ 30.00

Integral Quality
P.O. Box 31970
Seattle, Washington 98103-0070
(206) 527-2918

Washington State residents add sales tax.
VISA and MASTERCARD accepted.
Shipping included for prepaid orders.

Magic Mushroom for the Alice (Listing Continued, text begins on page 58)

Listing One

```
740  EOL = LL + 1: RETURN
750  REM -----------------------------------------------------------
760  REM                         PAGINATION
770  REM -----------------------------------------------------------
780  FILL = 66 - BM - LINES : PRINT NLF$;
790  IF FILL > 0 THEN  FOR L = 1 TO FILL: PRINT : NEXT
800  PRINT : PRINT : CENTER = LM + LL / 2 - 3
810  PRINT  SPC( CENTER);"- ";PG;" -": FOR L = 1 TO 6: PRINT : NEXT
820  FOR L = 1 TO TM: PRINT : NEXT
830  LINES = TM:PG = PG + 1: RETURN
840  REM -----------------------------------------------------------
850  REM          INITIALIZE VARIABLES AND DIMENSION ARRAYS
860  REM -----------------------------------------------------------
870  VTAB 10: HTAB 12: PRINT "INITIALIZING"
880  TM = 4:BM = 9:LINES = TM:PG = 1: REM TOP AND BOTTOM MARG, PAGE
890  LM = 10:RM = 70: LL = RM - LM: REM MARGINS, LENGTH OF LINE
900  CDA$ = "+": ENG$ = "^": REM COEUR D'ALENE AND ENGLISH FLAGS
910  FLAG$ = "+": REM INIT TO COEUR D'ALENE
920  DIM T$(LL): DIM B$(LL): REM TOP- AND BOTTOM-OF-LINE SECTORS
930  REM -----------------------------------------------------------
940  REM                       GRAPHICS FONTS
950  REM -----------------------------------------------------------
960  DIM G$(13): REM FONTS
970  GC$ =  CHR$ (27) +  CHR$ (76) +  CHR$ (12) +  CHR$ (0): REM
        GRAPHICS COMMAND
980  FOR M = 1 TO 13: REM READ FONTS
990     FOR N = 1 TO 12: REM READ COLUMNS
1000       READ G
1010       G$(M) = G$(M) +  CHR$ (G)
1020    NEXT N
1030    G$(M) = GC$ + G$(M)
1040 NEXT M
1050 DATA 32,96,78,78,72,72,120,48,0,0,0,0: REM G$(1) GLOTTAL STOP
1060 DATA 4,46,42,42,42,42,42,62,28,8,0,0: REM G$(2) SCHWA
1070 DATA 0,0,0,0,32,62,62,2,2,0,0,0: REM G$(3) IOTA
1080 DATA 32,48,24,12,118,118,12,24,48,32,0,0: REM G$(4) EJEC PALATAL
1090 DATA 32,48,24,12,6,6,12,24,48,32,0,0: REM G$(5) PALATAL
1100 DATA 0,0,0,0,80,80,96,96,0,0,0,0: REM G$(6) EJECTIVE
1110 DATA 0,0,0,8,24,48,96,64,0,0,0,0: REM G$(7) STRESS
1120 DATA 62,62,32,32,33,33,33,33,63,30,0,0: REM G$(8) NASAL VELAR
1130 DATA 0,0,8,40,56,24,12,12,8,8,0,0: REM G$(9) ETH TOP SECTOR
1140 DATA 28,62,34,34,34,34,38,62,60,0,0,0: REM G$(10) ETH BOT SECT
1150 DATA 16,48,34,34,34,34,34,54,62,28,0,0: REM G$(11) OPEN "O"
1160 DATA 14,14,18,18,32,32,18,18,14,14,0,0: REM G$(12) LAX MD-CNT VL
1170 DATA 20,62,42,42,42,42,42,42,0,0,0,0: REM G$(13) LOW-MID "E"
1180 REM -----------------------------------------------------------
1190 REM                      PRINTER COMMANDS
1200 REM -----------------------------------------------------------
1210 CR$  =  CHR$ (13)
1220 EMS  =  CHR$ (27) +  CHR$ (69): REM  EMPHASIS
1230 OFF$ =  CHR$ (27) +  CHR$ (70): REM TURN OFF SUP/SUBSCRIPT
1240 TF$  =  CHR$ (27) +  CHR$ (65) +  CHR$ (4): REM 4 DOTS LINE-FEED
1250 BF$  =  CHR$ (27) +  CHR$ (65) +  CHR$ (8): REM 8 DOTS LINE-FEED
1260 NLF$ =  CHR$ (27) +  CHR$ (65) +  CHR$ (12): REM NORMAL LN-FEED
1270 RS$  =  CHR$ (27) +  CHR$ (64): REM RESET
1280 SS$  =  OFF$ +  CHR$ (27) + "S" +  CHR$ (0) +  CHR$ (119) +
             CHR$ (27) + CHR$ (72) + EM$: REM SUPERSCRIPT COMMAND
1290 BX$  =  CHR$ (27) + "-" +  CHR$ (1) +  CHR$ (120) + CHR$ (27) +
             "-" +  CHR$ (0): REM BARRED "X"
1300 LT$  =  CHR$ (108) +  CHR$ (8) +  CHR$ (126): REM "L" BKSP TILDA
1310 REM -----------------------------------------------------------
1320 REM                             DOS
1330 REM -----------------------------------------------------------
1340 D$ =  CHR$ (13) +  CHR$ (4)
1350 PRINT D$;"MONI"
1360 VTAB 10: HTAB 10: PRINT "TURN ON PRINTER"
1370 VTAB 11: HTAB 10: PRINT "INSERT TEXT DISKETTE"
1380 HTAB 10: PRINT "PRESS ANY KEY TO CONTINUE";
1390 GET Q$: HOME
1400 PRINT D$;"CATALOG"
1410 INVERSE : PRINT "RIGHT DISKETTE? Y OR N> ";
1420 GET Q$: IF Q$ = "Y" THEN  PRINT : GOTO 1460
1430 NORMAL : HOME : VTAB 12: HTAB 10
```

(Continued on page 70)

Magic Mushroom for the Alice

Listing One

(Listing Continued, text begins on page 58)

```
1440  PRINT "INSERT CORRECT DISKETTE"
1450  GOTO 1380
1460  INPUT "TEXT FILE NAME> ";T$: NORMAL
1470  GOSUB 1550: REM GET LINE-SPACING
1480  PRINT D$;"VERIFY ";T$
1490  PRINT D$;"OPEN ";T$
1500  PRINT D$;"PR#1";
1510  PRINT D$;"READ ";T$
1520  PRINT EM$;
1530  FOR L = 1 TO TM - 2: PRINT : NEXT : RETURN
1540  REM ----------------------------------------------------------
1550  REM                        LINE-SPACING
1560  REM ----------------------------------------------------------
1570  HOME : VTAB 8: HTAB 10: PRINT "DEFAULT SPACING> 0. RETURN"
1580  PRINT : HTAB 18: PRINT "SPACING> ";: INPUT S$
1590  IF S$ = "" THEN S = 0: RETURN
1600  IF  ASC (S$) < 48 OR  ASC (S$) > 57 THEN 1550
1610  S =  VAL (S$): RETURN
1620  REM ----------------------------------------------------------
1630  REM                        TERMINATION
1640  REM ----------------------------------------------------------
1650  IF  PEEK (222) = 5 THEN EOL = X + 1: GOSUB 410: GOSUB 760:
        PRINT RS$: PRINT D$;"PR#0": PRINT "END OF FILE": END
1660  IF  PEEK (222) = 6 THEN : HOME : VTAB 8: HTAB 12: INVERSE :
        PRINT "FILE NOT FOUND": FOR T = 1 TO 2000: NEXT : NORMAL :
        GOSUB 1380: GOSUB 650: GOTO 60
1670  PRINT RS$: PRINT D$;"PR#0"
1680  PRINT "DOS ERROR STATUS: "; PEEK (222): END
1690  REM ----------------------------------------------------------
1700  REM                  PHONETIC COEUR D'ALENE
1710  REM               COPYWRITE © ALL RIGHTS RESERVED
1720  REM
1730  REM            GARY B. PALMER, ASSOCIATE PROFESSOR
1740  REM
1750  REM          DEPT OF ANTHROPOLOGY AND ETHNIC STUDIES
1760  REM         UNIVERSITY OF NEVADA, LAS VEGAS, NV 89154
1770  REM
1780  REM                   DECEMBER 31, 1983
1790  REM ----------------------------------------------------------
1800  REM  PRINTS ENGLISH AND PHONETIC COEUR D'ALENE.
1810  REM  MIXES CHARACTER SETS IN SAME LINE.
1820  REM  TEXT CODE CHARS: "+" = CDA, "^" = ENG.
1830  REM  CREATES VERTICALLY EXTENDED FONTS BY SPLICING TWO ARRAYS.
1840  REM  DOES WORD-WRAP, PAGINATION, AND VARIABLE SPACING.
1850  REM  ALL LOWER CASE CHARS EXCEPT "J" AND "O" PASSED TO PRINTER.
1860  REM  UPPER CASE A,B,F,J,K,M,P,Q,V,Y,Z
1870  REM  SPECIAL CHARACTERS C,D,E,G,H,I,L,N,O,S,T,U,W,X,!,'
1880  REM  REQUIRES APPLE ][ PLUS AND EPSON MX-100.
1890  REM  MARGINS: 4 TOP, 6 BOTTOM, 10 LEFT, 70 RIGHT
```

End Listing One

Listing Two

```
t     IF A$ = "any char" THEN T$(X) = G$(M): B$(X) =
      G$(M+1): RETURN
      DATA 32,32,127,127,32,32,0,0,0,0,0,0: REM G$(M)
      DATA 0,0,12,14,2,2,2,0,0,0,0,0: REM G$(M+1)

d     IF A$ = "any char" THEN T$(X) = G$(M): B$(X) =
      G$(M+1): RETURN
      DATA 0,2,2,2,2,2,2,126,126,0,0,0: REM G$(M)
      DATA 28,30,2,2,2,2,2,30,31,1,1,0: REM G$(M+1)

J     IF A$ = "any char" THEN T$(X) = G$(M): B$(X) =
      G$(M+1): RETURN
      DATA 0,2,2,62,62,0,0,0,0,0,0,0: REM G$(M)
      DATA 28,28,4,28,24,0,0,0,0,0,0,0: REM G$(M+1)
```

(Continued on page 72)

ENVOY™

Communications Software

- Easy to use, menu driven, compact and high speed
- Access electronic mail, remote systems and data networks like CompuServe, The Source and Dow Jones News Retrieval
- Terminal mode with large data-capture buffer
- Error-free transfers of text and binary files
- XMODEM (Christensen) and ANSI X3.28 transfer protocols
- User-definable Autodial and Autologin menu
- Utilities menu for file copy, type, print, erase and rename
- Remote unattended file transfers and utilities
- Available for most popular computer systems: IBM PC, PCjr, PC compatibles, Sanyo MBC-55X, MSDOS, CP/M-86, Concurrent CP/M, CP/M 2.2, CP/M Plus and more
- Money-back guarantee if not completely satisfied

49^{95}

ARTISOFT INC

P.O. Box 41436
Tucson, AZ 85717
(602) 327-4305

Circle **no. 7** on reader service card.

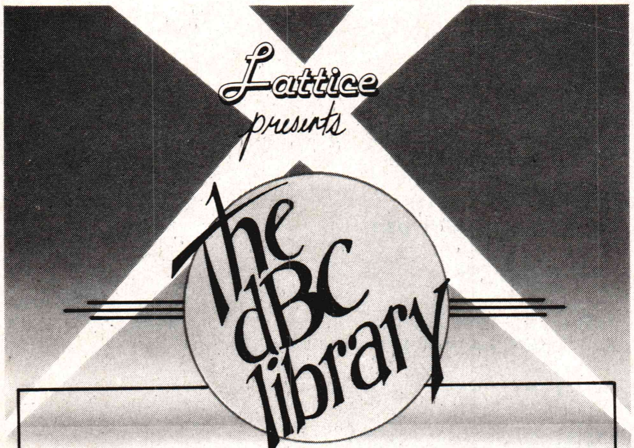
Electronic Circuit Analysis

- New release
- Transient, AC, DC analysis
- Full nonlinear
- Over 200 nodes
- Full editing
- Macro circuits
- Worst case, Monte-Carlo
- Temperature effects
- Frequency dependent parts
- Time dependent parts

For MS-DOS. 192k minimum.
$395.00

Tatum Labs
33 Main Street
Newtown, CT 06470
(203) 426-2184

Circle **no. 82** on reader service card.



The new Lattice dBC Library commands great performances between C language programs and dBASE files.

With the dBC software tool kit, you can extend any existing dBASE-II or dBASE-III application, or write complete new applications. dBC provides 37 functions that easily add, update, delete, retrieve and organize records as well as their corresponding indexes. And up to 8 data and 8 index files can be opened and processed simultaneously.

The dBC Library accesses dBASE files under the MS-DOS, PC-DOS and UNIX environments.

It operates with popular C compilers such as Lattice C, CI-C86 and DeSmet C88. No object license is required to re-sell products developed with the dBC Library.

Put the spotlight on your dBASE programming...order today!

PRICE: $250 per copy ($500 with source code)

ASK ABOUT OUR "TRADE UP TO LATTICE C POLICY"

For more information or to place your order contact:

LATTICE®
P.O. Box 3072
Glen Ellyn, IL 60138
(312) 858-7950
TWX 910-291-2190

dBASE is a trademark of Ashton-Tate
dBC is a trademark of Lattice

Circle **no. 58** on reader service card.

Only $95 with FULL SOURCE CODE!

Q/C

"... an incredible learning tool." *Byte*

For only $95, Q/C is a ready-to-use C compiler for CP/M with complete source code. Here's what *BYTE* (May 1984) said: "Q/C ... has a portable library and produces good code quality. If you want to learn compiler construction techniques or modify the standard language, Q/C is the obvious choice."

- Source code for compiler and over 75 library functions.
- Strong support for assembly language and ROMs.
- No license fees for object code.
- Z80 version takes advantage of Z80 instructions.
- Q/C is standard. Good portability to UNIX.

Q/C has casts, typedef, sizeof, structure initialization, and function typing. It is compatible with UNIX Version 7 C, but doesn't support long integers, float, parameterized #defines, or bit fields. Call about our new products: Q/C profiler, Z80 code optimizer, and Z80 assembler and virtual linker, all with full source code!

THE CODE WORKS

5266 Hollister, Suite 224
Santa Barbara, CA 93111
(805) 683-1585

Q/C, CP/M, Z80, and UNIX are trademarks of Quality Computer Systems, Digital Research, Zilog, Inc., and Bell Laboratories respectively.

Circle **no. 6** on reader service card.

Magic Mushroom for the Alice (Listing Continued, text begins on page 58)

Listing Two

```
ʔ    IF A$ = "any char" THEN B$(X) = G$(M): RETURN
     DATA 32,96,78,78,72,72,120,48,0,0,0,0: REM G$(M)
     (See also, Listing 1, lines 240, 1050)

ɱ    IF A$ = "any char" THEN B$(X) = G$(M): RETURN
     DATA 28,28,32,32,28,28,33,33,31,30,0,0: REM G$(M)

ɳ    IF A$ = "any char" THEN B$(X) = G$(M): RETURN
     DATA 62,62,32,32,32,32,62,31,1,1,3,0: REM G$(M)

ɲ    IF A$ = "any char" THEN B$(X) = G$(M): RETURN
     DATA 1,1,63,62,32,32,32,32,62,30,0,0: REM G$(M)

ŋ    IF A$ = "any char" THEN B$(X) = G$(M): RETURN
     DATA 62,62,32,32,33,33,33,33,63,30,0,0: REM G$(M)
     (See also Listing 1, lines 350, 1120)

ɼ    IF A$ = "any char" THEN B$(X) = CHR$(108) + CHR$(8)
     + CHR$(126)
     (See also, Listing 1, lines 280, 1300)

ɦ    IF A$ = "any char" THEN T$(X) = G$(M): B$(X) =
     G$(M+1): RETURN
     DATA 30,30,2,2,2,2,2,0,0,0,0,0: REM G$(M)
     DATA 30,30,0,0,0,0,25,25,15,6,0.0: REM G$(M+1)

ɭ    IF A$ = "any char" THEN T$(X) = G$(M): B$(X) =
     G$(M+1): RETURN
     DATA 64,64,127,127,0,0,0,0,0,0,0,0: REM G$(M)
     DATA 0,0,30,30,2,2,0,0,0,0,0,0: REM G$(M+1)

ʎ    IF A$ = "any char" THEN B$(X) = G$(M): RETURN
     DATA 2,6,12,24,56,108,70,66,0,0,0,0: REM G$(M)

ɾ    IF A$ = "any char" THEN B$(X) = G$(M): RETURN
     DATA 2,2,30,62,34,34,50,16,0,0,0,0: REM G$(M)

ɽ    IF A$ = "any char" THEN T$(X) = G$(M): B$(X) =
     G$(M+1): RETURN
     DATA 63,63,16,48,32,32,32,32,32,0,0,0: REM G$(M)
     DATA 14,15,1,1,3,3,0,0,0,0,0,0: REM G$(M+1)

ɸ    IF A$ = "any char" THEN T$(X) = G$(M): B$(X) =
     G$(M+1): RETURN
     DATA 6,6,40,40,62,62,40,40,6,6,0,0: REM G$(M)
     DATA 16,16,10,10,30,30,10,10,16,16,0,0: REM G$(M+1)

β    IF A$ = "any char" THEN T$(X) = G$(M): B$(X) =
     G$(M+1): RETURN
     DATA 30,30,16,16,16,18,30,12,0,0,0,0: REM G$(M)
     DATA 31,31,5,4,4,4,28,24,0,0,0,0: REM G$(M+1)

θ    IF A$ = "any char" THEN T$(X) = G$(M): B$(X) =
     G$(M+1): RETURN
     DATA 31,63,34,34,34,34,34,34,63,31,0,0: REM G$(M)
     DATA 12,14,2,2,2,2,2,2,14,12,0,0: REM G$(M+1)

ð    IF A$ = "any char" THEN T$(X) = G$(M): B$(X) =
     G$(M+1): RETURN
     DATA 0,0,8,40,56,24,12,12,8,8,0,0: REM G$(M)
     DATA 28,62,34,34,34,34,38,62,60,0,0,0: REM G$(M+1)

ɹ    IF A$ = "any char" THEN B$(X) = G$(M): RETURN
     DATA 2,2,2,2,2,6,4,62,62,0,0,0: REM G$(M)

ɐ    IF A$ = "any char" THEN T$(X) = G$(M): B$(X) =
     G$(M+1): RETURN
     DATA 12,12,10,10,10,10,12,4,0,0,0,0: REM G$(M)
     DATA 62,63,41,41,41,40,56,16,0,0,0,0: REM G$(M+1)

ʑ    IF A$ = "any char" THEN B$(X) = G$(M): RETURN
     DATA 68,76,92,116,102,71,1,1,1,0,0,0: REM G$(M)

ʃ    IF A$ = "any char" THEN T$(X) = G$(M): B(X) =
```

```
          G$(M+1): RETURN
          DATA 0,0,0,6,14,24,16,16,0,0,0,0: REM G$(M)
          DATA 1,1,3,30,28,0,0,0,0,0,0,0: REM G$(M+1)

ʒ         IF A$ = "any char" THEN B$(X) = G$(M): RETURN
          DATA 108,108,66,66,82,114,108,76,0,0,0,0: REM G$(M)

ɾ         IF A$ = "any char" THEN B$(X) = G$(M): RETURN
          DATA 1,29,63,34,34,34,34,34,34,0,0,0: REM G$(M)

ʐ         IF A$ = "any char" THEN B$(X) = G$(M): RETURN
          DATA 1,1,33,35,38,46,58,50,34,2,0,0: REM G$(M)

ç         IF A$ = "any char" THEN B$(X) = G$(M): RETURN
          DATA 56,125,69,69,71,71,68,68,0,0,0,0: REM G$(M)

ɣ         IF A$ = "any char" THEN T$(X) = G$(M): B$(X) =
          G$(M+1): RETURN
          DATA 32,48,25,15,6,6,15,25,48,32,0,0: REM G$(M)
          DATA 0,0,12,14,2,2,28,12,0,0,0,0: REM G$(M+1)

ħ         IF A$ = "any char" THEN B$(X) = G$(M): RETURN
          DATA 32,32,126,126,40,40,40,40,14,14,0,0: REM G$(M)

ʕ         IF A$ = "any char" THEN B$(X) = G$(M): RETURN
          DATA 48,120,72,72,78,78,96,32,0,0,0,0: REM G$(M)

ɦ         IF A$ = "any char" THEN B$(X) = G$(M): RETURN
          DATA 62,126,72,72,72,104,46,14,0,0,0,0: REM G$(M)

ɥ         IF A$ = "any char" THEN B$(X) = G$(M): RETURN
          DATA 112,112,16,16,16,16,127,127,1,1,0,0: REM G$(M)

ʋ         IF A$ = "any char" THEN B$(X) = G$(M): RETURN
          DATA 56,60,6,2,34,38,60,56,0,0,0,0: REM G$(M)

ʁ         IF A$ = "any char" THEN B$(X) = G$(M): RETURN
          DATA 126,126,18,18,18,18,18,126,108,0,0,0:
          REM G$(M)

ɨ         IF A$ = "any char" THEN T$(X) = G$(M): B$(X) =
          G$(M+1): RETURN
          DATA 0,0,24,24,0,0,0,0,0,0,0,0: REM G$(M)
          DATA 16,18,62,62,18,16,0,0,0,0,0,0: REM G$(M+1)
```

All the remaining vowels are produced entirely within the bottom line, as in the barred-u character which follows:

```
ʉ         IF A$ = "any char" THEN B$(X) = G$(M): RETURN
          DATA 16,16,60,62,18,18,18,62,60,16,16,0: REM G$(M)

ʌ         DATA 28,29,35,38,46,58,50,98,92,28,0,0

p         DATA 28,28,34,34,62,62,42,42,46,44,0,0

ɔ         DATA 16,48,34,34,34,34,34,54,62,28,0,0

ɒ         DATA 62,62,20,54,34,34,34,34,62,28,0,0

ø         DATA 20,62,42,42,42,42,42,42,0,0,0,0

æ         DATA 36,46,42,42,42,62,62,42,42,46,44,0

ə         DATA 4,46,42,42,42,42,42,62,28,8,0,0

ɐ         DATA 32,60,30,42,42,42,42,42,58,18,0,0

ɯ         DATA 60,60,2,2,60,60,2,2,60,62,2,0

ɤ         DATA 44,62,18,18,62,44,0,0,0,0,0,0

ʌ         DATA 14,14,18,18,32,32,18,18,14,14,0,0

ɑ         DATA 28,62,34,34,34,34,54,20,62,62,0,0
```

End Listings

Let's Mouse Around

A Turbo Pascal/Microsoft Mouse Sketching Program for IBM PCs

by Vincent and
Ronald G. Parsons

The program in the listing (page 75) is fun to use and illustrates techniques for using the Microsoft Mouse with Turbo Pascal. The program requires the IBM Color/Graphics adapter, a Microsoft Mouse, and Turbo Pascal.

By moving the mouse around, anyone can be an artist. When the program starts, an X-shaped cursor appears in the middle of a blue screen. A line is drawn on the screen as the mouse is moved with either mouse button depressed.

Below the drawing area is a set of options for changing colors, erasing portions of the screen, clearing the screen, and quitting the program. If the mouse cursor is placed between the [ ] after an option, the option is executed. Choosing "Back" changes the background color, "Palette" selects another palette of colors for drawing, and "Color" changes the color of the lines to be drawn. The "Eraser" option changes the cursor to a square and erases the portion of the drawing area in the square when a button is down. The "Clear" option clears the drawing area and "Quit" ends the drawing session.

Some of the more interesting techniques for using the Microsoft Mouse with Turbo Pascal are discussed below.

The Microsoft Mouse function 9 (Set Graphics Cursor Block) allows the program to redefine the mouse cursor shape and the position of the "hot spot" of the cursor. The hot spot is the position relative to the upper left corner of the 16 × 16 pixel cursor block where the position of the cursor is defined. The call to this function and function 12 (Set User-Defined Subroutine Input Mask) requires slightly different parameters than calls to the other mouse functions. The fourth parameter of functions 9 and 12 is an offset of an array and subroutine, respectively. The segment address of this array or subroutine must be passed to the mouse interrupt routine in register ES. We elected to make the segment address a global variable to keep the procedure parameters for the rest of the functions unchanged.

If you want to define a new cursor shape, a screen and cursor mask must be defined and placed in two contiguous arrays in storage and a pointer to these masks passed to function 9. Several different cursor shapes are defined in the Microsoft Mouse manual. The examples are given for BASIC. However, the storage order of arrays is different in Turbo Pascal than it is in BASIC. Thus, the two-dimensional array defined in the listing has the parameters in reverse order from the equivalent BASIC arrays in the mouse document. The horizontal and vertical hot spot locations are passed in m3 and m4, respectively. The hot spot can be within the range of −16 to 16 from the upper left corner of the cursor block.

When you use medium resolution

A mouse makes drawing with a computer easy. Here's a program that makes using a mouse easy.

Vincent and Ronald G. Parsons, 9001 Laurel Grove Dr., Austin, TX 78758.

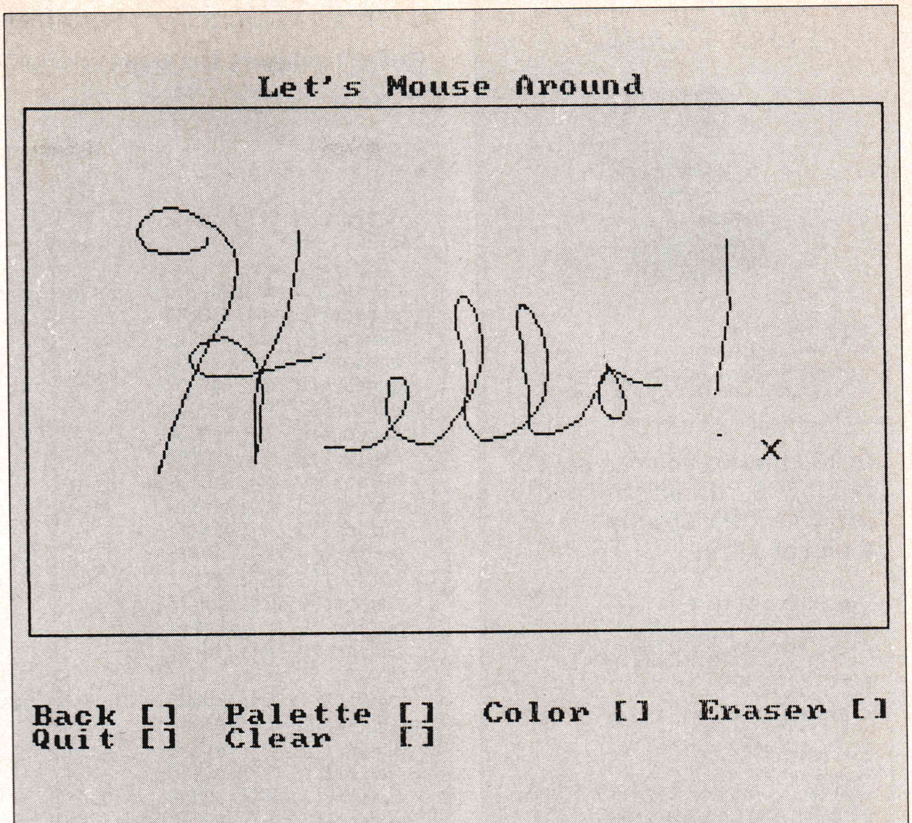
graphics mode with Turbo Pascal and the mouse, the x coordinate of the mouse must be divided by two before the Turbo Pascal Plot or Draw procedures can use it. Thus, the x coordinate is calculated as

x := m3 div 2;

The cursor must be hidden before you modify any portion of the screen containing the cursor. Mouse function 2 (Hide Cursor) performs this duty. If this is not done, the portion of the modified screen behind the cursor block will disappear when the cursor is moved. Functions 1 and 2 must be used in pairs because these functions increment and decrement a cursor flag and the cursor is displayed only when the cursor flag is zero. The initial value of the flag is −1.

Turbo Pascal permits fast development of programs, and the Microsoft Mouse makes the programs easier to use. DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service No. 196.



Figure

Let's Mouse Around Listing

(Listing Continued, text begins on page 74)

```
{ Mouse_sketch routine for Turbo Pascal and Microsoft Mouse }
{ Uses redefined cursor and Medium resolution graphics in color }
{  by Ronald G. Parsons and Vincent L. B. Parsons }

Program Mouse_sketch;

type regpack = record
                 ax,bx,cx,dx,bp,si,di,ds,es,flags : integer;
               end;
     cursordef = array[0..1,0..15] of integer;

var  oldx,oldy,x,y : integer;
     i,esreg,m1,m2,m3,m4 : integer;
     cursor1,cursor2 : cursordef;
     background,palettenum,color : integer;
     exit,eraser : Boolean;

procedure mouse (var m1,m2,m3,m4:integer); {Microsoft Mouse interrupt routine}
var regset : regpack;
begin
  with regset do
  begin
    ax := m1;      {Set registers for mouse interrupt}
    bx := m2;
    cx := m3;
    dx := m4;
    if ((m1=9) or (m1=12)) then es:=esreg; {Set ES for functions 9 and 12}
  end;
  intr(51,regset); {Perform mouse interrupt 51}
  with regset do
  begin
    m1 := ax;      {Set parameters on return}
    m2 := bx;
    m3 := cx;
    m4 := dx;
```

(Continued on next page)

Let's Mouse Around Listing

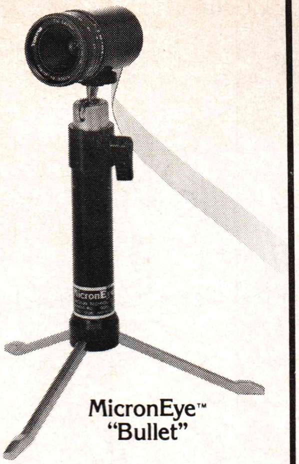
(Listing Continued, text begins on page 74)

```
    end;
end;

procedure initialize;
begin

    cursor1[0, 0]:=$07e0;  {Screen mask} {Diagonal Cross}
    cursor1[0, 1]:=$0180;
    cursor1[0, 2]:=$0000;
    cursor1[0, 3]:=$c003;
    cursor1[0, 4]:=$f00f;
    cursor1[0, 5]:=$c003;
    cursor1[0, 6]:=$0000;
    cursor1[0, 7]:=$0180;
    cursor1[0, 8]:=$07e0;
    cursor1[0, 9]:=$ffff;
    cursor1[0,10]:=$ffff;
    cursor1[0,11]:=$ffff;
    cursor1[0,12]:=$ffff;
    cursor1[0,13]:=$ffff;
    cursor1[0,14]:=$ffff;
    cursor1[0,15]:=$ffff;

    cursor1[1, 0]:=$0000;  {Cursor mask}
    cursor1[1, 1]:=$700e;
    cursor1[1, 2]:=$1c38;
    cursor1[1, 3]:=$0660;
    cursor1[1, 4]:=$03c0;
    cursor1[1, 5]:=$0660;
    cursor1[1, 6]:=$1c38;
    cursor1[1, 7]:=$700e;
    cursor1[1, 8]:=$0000;
    cursor1[1, 9]:=$0000;
    cursor1[1,10]:=$0000;
    cursor1[1,11]:=$0000;
    cursor1[1,12]:=$0000;
    cursor1[1,13]:=$0000;
    cursor1[1,14]:=$0000;
    cursor1[1,15]:=$0000;


    cursor2[0, 0]:=$0000;  {Screen mask} {Eraser}
    cursor2[0, 1]:=$0000;
    cursor2[0, 2]:=$0000;
    cursor2[0, 3]:=$1ff8;
    cursor2[0, 4]:=$1ff8;
    cursor2[0, 5]:=$1ff8;
    cursor2[0, 6]:=$1ff8;
    cursor2[0, 7]:=$0000;
    cursor2[0, 8]:=$0000;
    cursor2[0, 9]:=$0000;
    cursor2[0,10]:=$ffff;
    cursor2[0,11]:=$ffff;
    cursor2[0,12]:=$ffff;
    cursor2[0,13]:=$ffff;
    cursor2[0,14]:=$ffff;
    cursor2[0,15]:=$ffff;

    cursor2[1, 0]:=$0000;  {Cursor mask}
    cursor2[1, 1]:=$7ffe;
    cursor2[1, 2]:=$4002;
    cursor2[1, 3]:=$4002;
    cursor2[1, 4]:=$4002;
    cursor2[1, 5]:=$4002;
    cursor2[1, 6]:=$4002;
    cursor2[1, 7]:=$4002;
    cursor2[1, 8]:=$7ffe;
    cursor2[1, 9]:=$0000;
    cursor2[1,10]:=$0000;
    cursor2[1,11]:=$0000;
    cursor2[1,12]:=$0000;
    cursor2[1,13]:=$0000;
```

(Continued on page 79)

Now Your Computer Can See!

$295.00*



MicronEye™ "Bullet"

A total imaging system complete and ready for plug-and-go operation with your personal computer.

The MicronEye™ offers selectable resolution modes of 256 x 128 and 128 x 64 with operating speeds up to 15 FPS. An electronic shutter is easily controlled by software or manual functions, and the included sample programs allow you to continuously scan, freeze frame, frame store, frame compare, print and produce pictures in shades of grey from the moment you begin operation.

Only the MicronEye™ uses the revolutionary IS32 OpticRAM™ image sensor for automatic solid state image digitizing, with capability for grey-tone imaging through multiple scans. And with these features, the MicronEye™ is perfectly suited for graphics input, robotics, text and pattern recognition, security, digitizing, automated process control and many other applications.

The MicronEye™ is available with immediate delivery for these computers: Apple II, IBM PC, Commodore 64 and the TRS-80CC (trademarks of Apple Computer Inc., International Business Machines, Commodore Corp., and Tandy Corp. respectively).

Phone for MicronEye™ information on the Macintosh, TI PC and RS232 (trademarks of Apple Computer Inc. and Texas Instruments respectively.)

*(Add $10.00 for shipping and handling [Federal Express Standard Air]; residents of the following states *must add sales tax:* AK, AZ, CA, CO, CT, FL, GA, IA, ID, IL, IN, LA, MA, MD, ME, MI, MN, NC, NE, NJ, NY, OH, PA, SC, TN, TX, UT, VA, VT, WA, WI.)

MICRON TECHNOLOGY, INC.
SYSTEMS GROUP
1475 Tyrell Lane
Boise, Idaho 83706
(208) 386-3800
TWX 910-970-5973

Circle **no. 60** on reader service card.

FINALLY... THE C JOURNAL

The C Journal will help YOU use C on YOUR machine —IBM PC™, CP/M™, Macintosh™, or UNIX™-based — micro, mini, or mainframe.

It's the ONE publication for programmers, software managers, and other computer professionals who need to keep aware of developments in the industry's fastest-growing language.

- regular columns for novice through advanced C programmers
- software product and book reviews
- tips on working with major compilers and operating systems
- news from the ANSI standards committee and the industry

For **FREE** sample issue and discount subscription information, write, call, or circle our reader service number. **The C Journal** is a quarterly publication, and costs $28/year (add $9 for overseas airmail).

Subscriptions/Advertising:
Christina Gardner
(201) 989-0570

Editorial:
Rex Jaeschke
(703) 860-0091

another independent publication from
InfoPro Systems
3108 Route 10
P.O. Box 849
Denville, NJ 07834

MasterCard VISA

Circle **no. 91** on reader service card.

The Tools You Need To C You Thru.

Now the *WizardWare*™ Applications Programmers Toolkit provides everything you need to increase your C programming productivity.

APT™ features include:

- **COMPLETE SOURCE CODE (over 5000 lines!)**
- **File handling with direct & keyed access**
- **Screen and Report Generators, with full screen handling for your programs**
- **Generic Terminal Driver for portable code**
- **String math functions, and string manipulation routines**
- **Reference Manual on Disk (over 50 pages)**
- **Tutorial Manual (over 25 pages) with Source for Mailing List Manager**
- **A host of useful Utilities, Database and File Editors**
- **Available for Lattice C, Mark Williams C, DeSmet C, BDS C, others.**

Also Available: C-STARTER Toolkit, great for learning C!! Includes: Customized APT, DeSmet C Compiler, and "Programming in C on the IBM-PC" (200 pages)

APT/MS-DOS versions	$495
APT/DeSmet C version	$395
APT/BDS C version	$395
C-Starter (binary APT, DeSmet Compiler and Book)	$295
APT/Manual only	$ 50

Detailed Brochures on request

*Manual Cost will be applied if APT purchased within 30 days ($10 re-stocking charge.) U.S. funds only, please.

Trademarks: MS-DOS/Microsoft, Lattice C/Lattice, Inc., MWC86/Mark Williams Co., DeSmet C/C Ware, CI-C86/Computer Innovations, Inc., BDS C/BD Software, DR-C/Digital Research, WizardWare, APT, C-Starter/Shaw ☆ American Technologies.

Call (502) 583-5527

Ask for APT™ or C-Starter, or Send Check to:

Shaw ☆ American Technologies
WizardWare™
830 South Second St. - Box 648
Louisville, KY 40201, USA

VISA master charge

(C.O.D. and Foreign Orders - Add $5 Shipping/Handling)
References: Bank of Louisville, Citizens Fidelity Bank, Louisville Chamber of Commerce

Circle **no. 86** on reader service card.

(LISP)

Artificial Intelligence Language UO-LISP Programming Environment The Powerful Implementation of LISP for MICRO COMPUTERS

LEARN LISP System (LLS.1) (see description below) **$39.95**

UO-LISP Programming Environment Base Line System (BLS.1) **$49.95**

Includes: Interpreter, Compiler, Structure Editor, Extended Numbers, Trace, Pretty Print, various Utilities, and Manual with Usage Examples. **(BLS.1)** expands to support full system and products described below.

UO-LISP Programming Environment: The Usual LISP Interpreter Functions, Data Types and Extensions, Structure & Screen Editors, **Compiler, Optimizer, LISP & Assembly Code Intermixing**, Compiled Code Library Loader, I/O Support, Macros, Debug Tools, Sort & Merge, On-Line Help, Other Utility Packages, Hardware and Operating System Access, Session Freeze and Restart, Manual with Examples expands to over 350 pages. Other UO-LISP products include: LISPTEX text formatter, LITTLE META translator writing system, RLISP high level language, NLARGE algebra system. Prices vary with configurations beyond (BLS.1) please *send for FREE catalog*.

LEARN LISP System (LLS.1): Complete with LISP Tutorial Guide, Editor Tutorial Guide, System Manual with Examples, Full LISP Interpreter, On-Line Help and other Utilities. LEARN LISP fundamentals and programming techniques rapidly and effectively. This system does not permit expansion to include the compiler and other products listed above.

LISP Tutorial Support (LTS.1): Includes LISP and Structure Editor Tutorial Guides, On-line Help, and History Loop. This option adds a valuable learning tool to the UO-LISP Programming Environment (BLS.1). *Order (LTS.1) for* **$19.95.**

REQUIRES: UO-LISP Products run on most Z80 computers with CP/M, TRSDOS or TRSDOS compatible operating systems. The 8086 version available soon.

TO ORDER: Send Name, Address, Phone No., Computer Type, Disk Format Type, Package Price, 6.5% Tax (CA residents only), Ship & Handle fee of $3.00 inside U.S. & CN, $10 outside U.S., Check, Money Order, VISA and MasterCard accepted. With Credit Card include exp. date. Other configurations and products are ordered thru our *FREE catalog*.

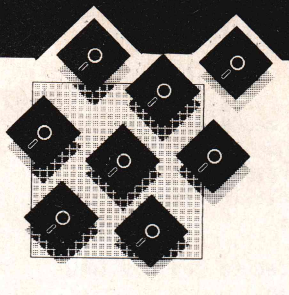
Northwest Computer Algorithms
P.O. Box 90995, Long Beach, CA 90809 (213) 426-1893

Circle **no. 61** on reader service card.

```
   cursor2[1,14]:=$0000;
   cursor2[1,15]:=$0000;

   background:=Blue;
   palettenum:=2;
   color:=3;
   GraphColorMode;
   GraphBackground(background); {Medium resolution graphics - Blue background}
   Palette(palettenum);
   eraser:=false;
   exit:=false;
   m1:=0;
   mouse(m1,m2,m3,m4);        {Reset mouse}
   m1:=1;
   mouse(m1,m2,m3,m4);        {Show cursor}
   m1:=9;
   m2:=7;                     {Horizontal hot spot}
   m3:=4;                     {Vertical hot spot}
   m4:=ofs(cursor1[0,0]);
   esreg:=seg(cursor1[0,0]); {Set pointer to cursor definition}
   mouse(m1,m2,m3,m4);        {Set graphics cursor}
   oldx:=160;                 {Center of screen}
   oldy:=100;

   gotoxy(1,1);
   write('          Let''s Mouse Around   ');
   gotoxy(1,23);
   write('Back []  Palette []  Color []  Eraser []');
   gotoxy(1,24);
   write('Quit []  Clear   []');
   draw(0,10,319,10,3);
   draw(319,10,319,169,3);
   draw(319,169,0,169,3);
   draw(0,169,0,10,3);
   GraphWindow(1,0,317,168); {Limit drawing window}
end; {initialize}

procedure changeback;
begin
  background:=background+1;
  if background>15 then background:=0;
  GraphBackground(background);
  delay(300);
end;

procedure changepalette;
begin
  palettenum:=palettenum+1;
  if palettenum>3 then palettenum:=0;
  palette(palettenum);
  delay(300);
end;

procedure changecolor;
begin
  color:=color+1;
  if color>3 then color:=0;
  delay(300);
end;

procedure makeeraser;
begin
  if eraser then
  begin
    m1:=9;
    m2:=7;                     {Horizontal hot spot}
    m3:=4;                     {Vertical hot spot}
    m4:=ofs(cursor1[0,0]);
    esreg:=seg(cursor1[0,0]); {Set pointer to cursor definition}
    mouse(m1,m2,m3,m4);        {Set graphics cursor}
  end
  else
  begin
```

```
      m1:=9;
      m2:=1;                        {Horizontal hot spot}
      m3:=1;                        {Vertical hot spot}
      m4:=ofs(cursor2[0,0]);
      esreg:=seg(cursor2[0,0]); {Set pointer to cursor definition}
      mouse(m1,m2,m3,m4);       {Set graphics cursor}
    end;
    eraser:=not eraser;
    delay(300);
  end;

procedure checkoption;
begin
  if m2 <> 0 then
  begin
    if ((m4 > 175) and (m4 < 183)) then
    begin
      if ((m3 >  81) and (m3 < 103)) then changeback;
      if ((m3 > 273) and (m3 < 295)) then changepalette;
      if ((m3 > 433) and (m3 < 455)) then changecolor;
      if ((m3 > 609) and (m3 < 631)) then makeeraser;
    end;
    if ((m4 > 183) and (m4 < 191)) then
    begin
      if ((m3 >  81) and (m3 < 103)) then exit:=true;
      if ((m3 > 273) and (m3 < 295)) then initialize;
    end;
  end;
end;

begin
  initialize;
  while not (keypressed or exit) do   {Exit when any key is pressed}
  begin
    m1:=2;
    mouse(m1,m2,m3,m4);     {Hide cursor}
    m1:=3;
    mouse(m1,m2,m3,m4);     {Get mouse position}
{   if m2 <> 0 then begin gotoxy(1,1);write(m3,'  ',m4,'   '); end; }
    if ((m4<11) or (m4>175)) then checkoption else
    begin
      x:=m3 div 2;        {In medium resolution, x coord. must be divided by 2}
      y:=m4;
      if m2 <> 0 then
      begin
        if eraser then
        begin
          for i:=0 to 7 do draw(x+i,y,x+i,y+8,0);
        end
        else
        begin
          draw(oldx,oldy,x,y,color);  {Draw when mouse button is pressed}
        end;
      end;
    end;
    oldx:=x;
    oldy:=y;                    {Update old x/y values}
    m1:=1;
    mouse(m1,m2,m3,m4);     {Show cursor}
  end;

    TextMode(c80);
  end.
```

End Listing



Turbo Toolbox, Version 1.0

Company: Borland International, 4113 Scotts Valley Drive, Scotts Valley, CA 95066
Operating System: CP/M-80, CP/M-86, MSDOS, PCDOS
Price: $49.95
Circle Reader Service No. 143
Reviewed by Karl R. Kachigan

The Turbo Toolbox by Borland for users of Turbo Pascal can reduce development time and improve the portability of your code. It includes three tools: Turbo-Access, Turbo-Sort, and GINST. Turbo-Access is a data-filing system using B-trees that allows you rapid access to large amounts of data. Turbo-Sort is a data-sorting system that uses the Quicksort algorithm. GINST generates a general terminal installation program much like TINST for Turbo Pascal. The only restriction on the tools is that they are compatible only with Turbo Pascal, version 2.0 (or higher).

After using all the tools, I can comfortably say that any user of Turbo Pascal should be able to use these tools and find them helpful. At first, I expected the Turbo-Access system to be too complex for all but the most advanced programmers, but I now believe that even novice programmers can manage it once they go through the sample programs. GINST will be a great help to software developers who would like their programs to use Turbo's screen control features on any terminal.

The Toolbox is one disk containing 16 files, 10 of which are "toolbox" files and 6 sample programs or data files. Source code is provided for all tools and programs except the GINST-related files. GINST is the only compiled tool, probably because it modifies the Turbo Pascal runtime code. The 102-page typeset manual is much better than the Turbo Pascal, version 2.0 manuals—I found fewer typos and errors.

At $49.95, the Turbo Toolbox is less of a bargain than Turbo Pascal, but once you use one of the tools, you probably will find it worth the price. It is available on the same operating systems as Turbo Pascal: CP/M-80, CP/M-86, MSDOS, and PCDOS. In fact, I suspect that there is little or no difference between the MSDOS and PCDOS versions; they ran comfortably on both my HP-150 and IBM PC. I didn't test the CP/M versions, but because the manual makes no distinctions in the Toolbox's performance on different systems, I assume they function much the same as the MSDOS/PCDOS version.

These tools appear to be useful for both the hobbyist and the software distributor. Borland graciously has allowed developers to distribute and/or sell programs whose object code includes source code from Turbo-Access, Turbo-Sort, and the terminal installation program generated by GINST and to paraphrase the installation section from the Turbo Pascal reference manual when writing their documentation.

Turbo-Access

I'm not sure how many times I've had to create a data base to handle a lot of information, then had to add the necessary routines to find, add, delete, edit, and list the elements. Turbo-Access, a data base system that you can incorporate into your programs, is composed of four source files: ACCESS.BOX, GETKEY.BOX, ADDKEY.BOX, and DELKEY.BOX. These let you quickly create a good-sized, custom data base.

My initial fears that only an experienced programmer need delve into Turbo-Access arose when phrases like B-trees, index files, keys, pages, and tree height popped up in the tutorial section. The manual, however, explicitly states that you don't need to know how B-trees work—just how to use them. The tutorial gives a nice overview on B-trees, an approach that uses keys to find elements in its data base.

Turbo-Access is really an indexed sequential access method (ISAM) data base. For those unfamiliar with B-trees or ISAMs, these methods organize data in a manner that separately stores a unique key (like your street address) so that accessing for a specific element requires only a few searches of the key directory. This is a fast, efficient way to access an item from a large data base. In addition to your data file, you generate an index file that includes the keys organized in whatever manner you wish to treat the data. For example, if you want to create a mailing list, you could use an alphabetical list of the last names as your key. Or you may want to categorize these addresses by state or zip code—these too could be keys.

Just what is involved in using Turbo-Access? Of the four separate source files, you always must include ACCESS.BOX; it contains all the procedures and data structures to set up and maintain the data files and index files. You may include or omit the other three files as long as ACCESS.BOX comes first. GETKEY.BOX provides the searching routines for the keys in the index files and the data in the data files. ADDKEY.BOX lets you add keys and data to the index/data files, and DELKEY.BOX lets you delete keys and data from these files.

To start using Turbo-Access, first define a record that contains all the information you wish to store per element in your data base. By structuring your record to include several

items, you can use any of those items as keys in indexing your data. The key, limited to less than 256 characters, must be a string. Once you have defined this record, decide how much data you'll need to store and what you want your keys to look like. This will help you determine the constants that Turbo-Access will need to set up its data base. The six constants are data record size, key length, page size, order, page stack size, and max tree height. You've already figured out the first two. You simply guess the page size and page stack, given typical operating ranges for these. Order is half the page size, and you can compute max tree height easily, given the above numbers. I was surprised to see how little effort was needed here after you assume the suggested ballpark figures.

Finally, determine what you want to do with your data. You probably will want a complete program to find, add, delete, edit, and list the data; the sample programs for each of these tasks should get you going. The manual suggests that you include in your data record an integer variable that you can use for status. When your program stores a data element, you can set this status value to distinguish the element later from a deleted or unused element. This is helpful for recreating an index file that is corrupt—the manual even gives an example of how to do this. Corruption of index files is the biggest problem with ISAMs, so following this suggestion is a good idea.

The sample program BTREE.PAS is a fairly extensive mailing list program. It includes the capabilities of finding, adding, editing, deleting, and listing the data, with a nice menu-driven entry mode and function selection. BTREE.PAS nicely demonstrates the power of Turbo-Access. It is more extensive in scope than the sample programs that demonstrate the separate tasks of creating an index/data file and finding, adding, deleting, and listing the data. After I used these simple examples to put together my own little data base, the BTREE.PAS example showed me some nice enhancements.

What are the limitations of Turbo-Access? First, if you need to include all of your source files, it takes a while to compile. Second, be prepared while debugging your program to know what compiler directives the B-tree modules have set. Third, although a boolean variable "ok" is set after many operations to determine if a task was successful, if the task fails, you have little indication as to why. Any error that Turbo-Access can't handle becomes a fatal error and bombs you out of your program; you are left with only an indication of the error code, record number, and filename, generated by the TAiocheck routine. I suspect you could modify this routine to create a global error number that could be tested. However, it would be nice if the error code tests available in Turbo-Sort were available here so that the programmer could handle some of the disk-related errors (e.g., disk full or write-protected).

Fourth, the MakeFile routines used to create index and data files destroy any existing files that are using the same filenames. If you want to test automatically whether index and data files already exist, you must try to open them first. Finally, should you wish to use duplicate keys (and

you can instruct the Turbo-Access routines to do this), you must do additional searches to find the data you may want. Hence, unique keys will simplify your data base searches.

Turbo-Sort

Many times I've needed to sort a list of items, whether numbers or strings, in descending or ascending order. Turbo-Sort provides a simple, efficient method to do so. It uses the Quicksort algorithm, a faster approach than the time-honored bubble sort. It allows you to sort data from disk and/or memory and to send the results to disk and/or memory. Why is it so flexible? You get to write the input and output processing procedures, plus the procedure that determines the sort priority.

Turbo-Sort is composed of three parts: input, sort, and output. Borland provides the code to sort, while you provide (1) the input procedure Inp that puts the data into the sort processing memory, (2) the output procedure Outp that takes the sorted data and puts it somewhere else, and (3) the less function Less that determines if one element is less than another element. This sounds like a lot, but by copying the sample programs, you'll quickly be up and running.

The Inp procedure takes the data you wish to sort and puts it into sort memory. Turbo-Sort uses all of the existing heap space to store your data. If there is insufficient heap space (the program can sort up to 32,767 elements), Turbo-Sort does a virtual memory operation and partitions some of the data to disk. If your program uses the heap, it must allocate enough heap space for itself before you enter Turbo-Sort. The minimum amount of memory needed to sort is three times the item size (in bytes) or three times 128 bytes, whichever is greater–this isn't much space. The Inp and Outp procedures, in essence, take or get your data from the sort heap, which is managed by TurboSort.

The nice thing about being in control of the Less procedure is that you can sort on multiple keys, sort any type of item (numeric or alphabetic), and sort in ascending or descending order. The sample programs show descending sorts on numbers (i.e., quantity of an item, stock number, etc.). You can get ascending sorts by changing the < sign to > and recompiling. Likewise, if you wish to sort a data list in several different ways—say, by quantity then by name—you can follow the multiple key sample program and structure the Less procedure with a case statement. One thing to beware of: Turbo-Sort does require that each element of your data be at least two bytes long.

To use Turbo-Sort, you simply include the SORT.BOX file when compiling your program. Turbo-Sort forward declares the Inp and Outp procedures and the Less function. Should any errors occur, Turbo-Sort returns an error number indicating what is wrong. Of the six error codes, two give you disk-related errors indicating a read error or a file create error. This is much like the IoResult function of Pascal that lets you trap system errors. As mentioned before, I wish the Turbo-Access system had provided this.

GINST

This tool generates a terminal installation program; it isn't the installation program itself. It needs to know the name of your program so that it can patch the proper disk file. GINST produces three files: a code file, a data file, and a message file. Using this tool is simple. GINST asks you for your program name, then the name you want for the terminal installation program. That's it. To customize your program for a new terminal, just run your terminal installation program.

When running your terminal installation program, you'll notice that it looks similar to Turbo Pascal's TINST program. You select the terminal to use from a list, then have your code updated to reflect the cursor control sequences and screen editing commands of that specific terminal. You can also modify the cursor control and screen editing sequences, add new terminal definitions to the list, and delete selections from the list.

Although I heartily approve of this new tool, I do have a few complaints. First, Borland has provided an IN-

STALL.DOC text file that is similar to the step-by-step installation of the Turbo Pascal manual for use in your own manual or for distribution with your software. Unfortunately, the text is riddled with typos—as if someone hastily transcribed the instructions from the manual to disk but did no spelling checks. Second, GINST will not let you abort during the prompt asking for the new name of your installation program. It would be nice if a control-C were acknowledged here.

Third, on the MSDOS/PCDOS version, I couldn't make a program and its new installation program work on both my MSDOS machine and my IBM PC. If I compiled a program on my MSDOS machine then generated an installation program, it would work fine on that machine. But when I ported both object code files and the data files to my IBM PC and tried to run the installation program, oops! no IBM PC installation menu—just the MSDOS terminal selections. However, if I recompiled only my main program on the IBM PC then ran the installation program, voila! the IBM PC monitor selections appear. Obviously, GINST inspects some part of the Turbo Pascal runtime module to determine what compiler (MSDOS or PCDOS) was used then uses the appropriate terminal selections. It sure would be nice to have one object file portable to both the IBM's screen and a normal terminal on an MSDOS machine. I suspect, with the appropriate ANSI.SYS file under PCDOS 2.0 (or higher), I could add the IBM to my MSDOS terminal list, but that's not clean. That solution also doesn't do much for me if I'm still running PCDOS 1.1. Oh, well!

Finally, it would be nice to customize the GINST terminal data file for the terminals I want before I run it and generate my custom installation program. Unfortunately, if your favorite terminal (like my HP-150) doesn't appear on GINST's data file, you must add it to every installation program you generate.

Summary

As I mentioned at the start, many of us can use these tools. In general, I

found Turbo Toolbox fairly bug free and quite usable, except for the problems noted. The folks at Borland have produced another decent product.

SPSS/PC

Company: SPSS, Inc., 444 N. Michigan Ave., Chicago, IL 60611
Computer: IBM PC/XT, AT,, and compatibles
Price: $795.00
Circle Reader Service No. xx
Reviewed by Dr. Steven Anthony Sola

Even before buying my microcomputer, I dreamed of having a miniature megalith of an IBM 370 on my desk, at my command, and totally within my grasp. I wanted it all: every interesting language, every systems tool, and especially the big number-crunching programs like SPSS or BMDP. The operating systems, the languages, and the applications are here, and now SPSS/PC has arrived. But I have a new concern: How good is this micro product?

For many years now, Statistical Programs for the Social Sciences (SPSS) has been the standard package that most graduate students have used with mainframe resources to analyze their thesis or dissertation data. Although it has competition from BMDP and SAS, SPSS is more than holding its own. For example, in the corporate environment as represented by the annual Datamation survey of mainframe applications packages (March 1984), the SPSS batch system scored better than its group average in all categories. Perhaps the best test of quality is a measure of the program's reliability and validity. Can it be trusted to process the data of a thesis or a dissertation?

Through the kind cooperation of an associate, Dr. Jilisa Snyder, we were able to evaluate the SPSS/PC micro product directly against the SPSS mainframe version running at the State University of New York at Albany. First, we ran Jilisa's dissertation using an IBM PC as a terminal under the control of PC TALK III, uploading the data, interactively running the SPSS mainframe programs, and finally downloading the results. We then took samples of the most arduous statistical procedures found in the dissertation and attempted to run them using the SPSS/PC micro product. The control language required only minor adjustment, and all programs ran without mishap. Finally, we compared the output of each package using side-by-side windows

The results were spectacular and reassuring: out of over 156K of output examined, results were identical in every important detail to the mainframe version. Not only were the results accurate to their scientific precision, but they were exact even in their final imprecise digits to the mainframe version! The only exceptions occurred in the numeric representation of data plots, where the micro version rounded more accurately than the mainframe version. Basing our opinion on this excellent level of performance, we are reasonably assured that the SPSS/PC system is at least as valid and reliable as the SPSS mainframe version.

Of course, this conclusion must be tempered by a caveat. We assessed only a small proportion of the many possible procedures in the SPSS package, principally the multiple regression and stepwise multiple regression procedures. On the other hand, the data used was all too real! It contained numerous missing values, not only in the independent variables but also in the dependent variables. The subsamples were unequal in size and disproportionate. In short, the data was the messy and ill-behaved sort so typical of the social and behavioral sciences.

Is it possible to run all of the mainframe package on the micro? Unfortunately, no. SPSS/PC has been tailored or, more accurately, truncated. The most serious omission is any form of multivariate analysis of variance; the MANOVA procedure is missing. True, the complete univariate analysis of variance procedure ANOVA is present, but this procedure is far from sophisticated. Other procedures that compare groups are acceptable as far as they go, but MEANS, ONEWAY, and T-TEST simply can't compare with MANOVA in power and flexibility. Also missing is any form of canonical correlation, discriminant analysis, logistic regression, or time series analysis.

On the other hand, simple descriptive data analyses providing means, standard deviations, and the like, including plots, are well provided. Particularly noteworthy is the excellent implementation of cross-tabulation and hierarchical log-linear models for multi-way contingency tables. Although the provision for pearson correlation and nonparametric statistical techniques is adequate, there is no provision for rank-order correlation nor for partial correlation analysis. A hierarchical cluster analysis is nicely done, but the cluster memberships cannot be saved. This failure to save also plagues the factor scores in an otherwise exemplary procedure of FACTOR. Similarly, casewise residual scores cannot be written out in the multiple regression procedure.

Such truncation is unwarranted in this otherwise excellent product. The total number of variables is limited to 200. The total number of cases ostensibly is unlimited but, in fact, is limited by an overall maximum of 64K workspace for data. When I enquired as to why this might be so, the folks at SPSS muttered some mumbo-jumbo about the limitations of the current crop of Fortran compilers for the 8088. Come come, fellows! It should be possible to overcome this 64K curse. As an additional footnote, SPSS/PC uses the Microsoft Fortran compiler. While SPSS/PC makes use of a lot of overlays, it requires at least 320K and uses a maximum of 384K. Surely using another 256K of available storage on the IBM PC wouldn't hurt. It appears that SPSS is not taking full advantage of the IBM PC's resources, and that further optimization may lead to a better product.

Although it is possible to run SPSS/PC without an 8087 numeric coprocessor, it is silly to do so. After shelling out $795 for SPSS/PC, not spending the additional $200 or so for the coprocessor, which greatly speeds

things along, just isn't sensible. A hard disk is a requirement, not only for the procedures themselves and associated files, which take up approximately 3200K, but also for the temporary files used during a program run.

Transferring data out of SPSS/PC in the form of reports is helped along by a report writer; however, the line length is limited to 132 columns, and there is no printer configuration file to customize it or any of the save commands to your printer. Transferring files between a mainframe and SPSS/PC is greatly facilitated by KERMIT, a neat little data link included with the package. KERMIT can transfer some types of system files with error checking, as well as data files. Transferring files between your favorite spreadsheet, data base manager, or word processor and SPSS/PC is smooth as long as plain vanilla ASCII is your favorite flavor. If you have more exotic tastes, you will need to know how to cook in a programming language. Unfortunately, SPSS/PC provides no recipes for interfacing with their SPSS systems files.

SPSS/PC does provide the most elaborate, comprehensive, well thought out, and usable documentation available anywhere, mainframe or micro! The SPSS/PC documentation is specific to the micro user, and the built-in statistics guide is easy to follow and genuinely useful. Illustrations, examples, and reference material on each command and procedure abound. The typeset documentation, which weighs in at several pounds, is well organized, indexed, and reads easily. SPSS/PC comes with a nice tutorial on disk, along with a demo program. As if this were not enough, SPSS/PC also provides instantly available on-screen help when running interactively.

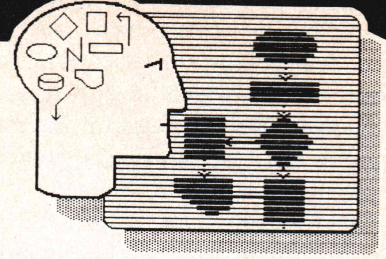
Yes, SPSS/PC can run interactively, with each command and procedure executed when entered. Traditional batch processing is also available. Unfortunately, SPSS/PC is severely hampered by the lack of an on-line editor. If you make a mistake, for example, in the middle of a long command line with many parameters, well, you had better enjoy typing! It is possible to get a directory listing and to examine or erase a DOS file, but a general exit and return to the operating system doesn't exist. The impact of this becomes more apparent when you realize that you must completely restart the whole package each time you use your favorite word processor to edit some commands.

Which brings us to the topic of the built-in copy protection and its insidious necessities. Every time you restart the program, you must have the "key diskette" in drive A. This is bad enough, but the SPSS ghouls are not through with you! At unpredictable times, in the middle of a procedure, the thing looks for the key in drive A. Woe to the foolish users who do not replace the key within a few tries into drive A, for their systems will come crashing down! Whatever you used to have in drive A may have vanished into bit heaven. For the typical type of system hang induced by these fiends involves keeping drive A running an infinite length of time. Can you use one of the bit copiers at least to provide yourself with some insurance? No, they don't appear to work. Will SPSS provide you with a backup disk? No way. Be prepared to wait about a week and to shell out more money when your key goes belly up (and they all eventually do). No, this isn't the middle of a nightmarish adventure game you've warped into—this is SPSS/PC's copy protection! (While we are on the subject of warnings: don't buy version 1.0. Version 1.1 corrects a number of significant oversights, and it will cost you an additional $100 to upgrade.)

Operationally, SPSS/PC purrs along like a kitten. It's fun and amusing to play with, and it appears immune to fatal operator errors. With an 8087 installed, it is quite frisky. Speed is not a major problem. SPSS/PC does what common sense and its manual says it will do, and it has been extensively debugged and carefully crafted. Support is provided by telephone during regular business hours (you pay the toll). Technical staff are knowledgeable, but you may have to wait 24 hours before they have time to fully answer your query. There is a mainframe-oriented users group but as yet no micro group. SPSS/PC will run on an IBM PC/XT, the AT, and on compatibles such as the Compaq, Tandy 2000, and TI Professional.

SPSS/PC is good, and it has brought me closer to my dream of having that IBM 370 on my desk. SPSS/PC's algorithms are fundamentally sound, it is finely crafted, and I would trust it to process a student's thesis. You won't find me in line to do batch processing nor cursing at the mainframe for being down again. I have my micro and I almost have it all!

DDJ



The Model in the Machine and the Model in the Mind

by Michael Swaine

How do you design software for people rather than for the machine or for the programmer? How do you make the model inside the machine reflect the model in the user's head? I cornered two programmers who think particularly hard about the intended users of software, and they talked about human factors in the processing of words and ideas.

Dave Winer is the president of Living Videotext; he wrote ThinkTank, the outline processor. While he devotes a lot of his time these days to meetings, budgets and strategic planning, he says, "what I am, first and foremost, is somebody who designs software." But he's a software designer who uses his own products and listens to others who use them.

Thom Hogan identifies himself here as a writer. As Editor-in-Chief of our sister magazine, *Business Software*, he's that, but he's also a programmer. It's his experience as a user of word-processing programs, though, that gave him his idea for exponential cursor movement.

Dave Winer: The first rule I try never to break is that if you don't use it yourself, it isn't going to be an interesting product. A lot of software designers don't use what they write. I guess we're fortunate in that our product, ThinkTank, is useful in a lot of different functions, so it's easy for us to say that everybody in the company uses the product. But you've got to use the product: we've modified ThinkTank many times over several years, and those modifications have been the result of heavy use of the product. Talk to any game designer about how he makes a game entertaining and he'll tell you the same thing: he plays the game himself, and he doesn't stop until it's entertaining for him.

But it's hard to put yourself in the user's frame of mind; you've got to continually cull out the things that make you too specialized. You have to work at seeing things the way a naive user sees them, and you have to know how to feel the little jolt when a program isn't right.

There's a term we use: burning brain cells. Every time the program does something that you're not expecting it to do, every time it surprises you just a little, your suspension of disbelief goes away. You're no longer thinking of the problem that you're working on, but about why the program misbehaved that way. It's that phenomenon of the software taking you away from the task at hand that we call burning brain cells.

The concept of suspension of disbelief is very important. In software just as in the movies or in fiction you have an audience, and you're trying to manipulate them, to influence their behavior; initially with a program you're trying to relax the person and invite him to try things. The suspension of disbelief develops as you draw the person in gradually: first he tries the cursor keys, say, and he finds that the cursor keys move the cursor; OK. Then what? Well, people make guesses; and you as a software designer have to try to anticipate their guesses. To a certain extent you do this on your own, but eventually you've got to try it out on real people. We found with ThinkTank that after the cursor keys, people were reaching for the spacebar, so we used the spacebar to step through the command line. So you second-guess the user, and if you do a good job, you build up a certain amount of trust on the part of the user; then over time, if it is going to be a successful experience for him, he will have suspended his disbelief. He's now no longer thinking about the software, he's thinking about the mechanics of the problem at hand. That's the goal.

One technique for getting there is not trying to force your structure on the user, but rather trying to get him to recognize his own structure in the machine. One of the reasons people like using computers is that it's neat to crawl around in there and discover things. Well it's neater if you're crawling around discovering things that you made rather than things that some guy in Cambridge made. We tried to capture a human structure in writing ThinkTank: categorizing and subcategorizing are very natural to the human mind, even though the way ThinkTank does categorizing is very different from the way humans do it. We've listened to users and in the next release incorporated features like cloning and hoisting and the use of color, all things that make the model closer to the model the user has.

Thom Hogan: I'm a writer. I spend a lot of time using word processors, and I've yet to find a word-processing program that doesn't somehow inhibit my writing and editing style. There are so many needed features of word processors that haven't been implemented yet that I sometimes wonder what to call the software I do use. This brief note describes just one concept that has yet to be implemented in any word processor.

Cursor control is out of control. I just got the latest whiz-bang word processor and I find that I can now move by character, by word, by phrase, by sentence, by paragraph, by block, by screen page, by actual page and by section. That's a *lot* of cursor keys.

Moving around in a document, especially a long one like a book or software manual, should not be a many-

THE PROGRAMMER'S SHOP™

helps compare, evaluate, find products. Straight answers for serious programmers.

SERVICES

- Programmer's Referral List
- Compare Products
- Help find a Publisher
- Evaluation Literature free
- BULLETIN BOARD - 7 PM to 7 AM 617-826-4086
- Dealer's Inquire
- Newsletter
- Rush Order
- Over 700 products

Free Literature - Compare Products

Evaluate products Compare competitors. Learn about new alternatives. One *free call* brings information on just about any programming need. Ask for any ***"Packet"*** or "Addon Packet": ☐ ADA, Modula ☐ "AI" ☐ BASIC ☐ "C" ☐ COBOL ☐ Editors ☐ FORTH ☐ FORTRAN ☐ PASCAL ☐ UNIX/PC or ☐ Debuggers, Linkers, etc.

RECENT DISCOVERIES

FASTER C - Lattice users eliminate Link step. Normal 27 seconds, Faster C in 13 secs. MSDOS $95

ARTIFICIAL INTELLIGENCE

EXSYS - Expert System building tool. Full RAM, Probability, Why, Intriguing, serious. PCDOS $275

GC LISP - "COMMON LISP", Help, tutorial, co-routines, compiled functions, thorough. PCDOS $475

IQ LISP - MACLISP & INTERLISP. Full RAM. Liked. PCDOS $155

TLC LISP - "LISP-machine"-like, all RAM, classes, turtle graphics 8087. CP/M-86, MSDOS $235

TLC LOGO - fast, classes. CPM $ 95

PROLOG-86 - Learn fast, Standard, tutorials, samples of Natural Language, Exp. Sys. MSDOS $125

Expert System front-ends for PROLOG: APES ($275), ES/P ($1895)

Other solid alternatives include: MuLISP-86 ($189), WALTZ LISP for CPM ($159), MicroPROLOG ($275)

EDITORS FOR PROGRAMMING

BRIEF Programmer's Editor - undo, windows, reconfigurable, macro programs, powerful. PCDOS $195

VEDIT - well liked, macros, buffers, CPM-80-86, MSDOS, PCDOS $119

MACINTOSH

We evaluate, carry every available programmers product. Ask.

C LANGUAGE

INSTANT C - Interactive development - Edit, Source Debug, run. Edit to Run - 3 Secs. MSDOS $ 495

"INTRODUCING C" - Interactive C to learn fast. 500 page tutorial, examples, graphics. PCDOS $ 95

MEGAMAX C - native Macintosh has fast compile, tight code, K&R, toolkit, .OBJ, DisASM MAC $ 295

CROSS COMPILERS by Lattice, CI. VAX to 8086. VMS $3000

C LIBRARIES

COMMUNICATIONS by Greenleaf ($149) or Software Horizons ($139) includes Modem7, interrupts, etc. Source. Ask for Greenleaf demo.

C SHARP Realtime Toolkit - well supported, thorough, portable, objects, state sys. Source MANY $ 600

PORTABLE C-LIB: Same calls for IBM, IIe, CP/M, C64, more. Screen, I/O, Graphic, more. $ 125

ROMPack - special $Main .EXE editor, source, tech support, 8086. $185

DEBUGGERS

PERISCOPE DEBUGGER - load after "bombs", symbolic, "Reset box", 2 Screen, own 16K. PCDOS $ 285

SOURCE PROBE by Atron for Lattice, MS C, Pascal. Windows single step, 2 screen, log file. $395

FORTRAN LANGUAGE

RM/FORTRAN - Full '77, big arrays. 8087, debugging, xref, MSDOS $525

DR/Fortran-77 - full ANSI 77, 8087, overlay, full RAM, big arrays, complex NUMS., CPM86, MSDOS $249

Ask about Microsoft, Supersoft, others.

OTHER LANGUAGES

ASSEMBLER - ask about Microsoft MASM-86 ($125) improvements or its new competitors.

"BASICA COMPILER": Better BASIC all RAM, modules, structure. $185

HS/FORTH - '79 & '83 Standards, full RAM, ASM, BIOS, interrupts, graph, multi-task, optimizer MSDOS $250

MBP COBOL has screen control, strong doc, '74 interm., fast. MSDOS $680

SUPPORT PRODUCTS

BASIC DEVELOPMENT SYSTEM - (BDS) for BASICA; Adds Renum, crossref, compress. PCDOS $115

PLINK-86 for Overlays, most lang., segment control. MSDOS $325

ProYAM Communications Package - All a programmer'd want. TTY, VT 100, 3101, MODEM7, BBS. Remote, macros, windows MSDOS $139

SCIL - Source Librarian to manage Versions, Doc, Minimize disk space, confusion. MSDOS $335

"C" LANGUAGE

	OUR PRICE
MSDOS: C86-8087, reliable	call
Instant C - Inter., fast, full	495
Lattice 2.1 - improved	call
Microsoft C 2.x	329
Williams, debugger, fast	call
C Systems & debugger	175
CPM80: Ecosoft C-now solid, full	225
BDS C - solid value	125
MACINTOSH: Softworks	365
Megamax-object, full	295
Consulair's MAC C	295

Compare, evaluate, consider other Cs

BASIC

	RUNS ON	
Active Trace-debug	86/80	75
BASCOM-86 - MicroSoft	8086	279
BASIC Dev't System	PCDOS	115
BetterBASIC - 640K	PCDOS	185
CB-86 - DRI	CPM86	419
Prof. BASIC Compiler	PCDOS	89
Databurst - screens	MSDOS	215
SCREEN SCULPTOR	PCDOS	115

Ask about ISAM, other addons for BASIC

SERVICE

ALL PRODUCTS - We carry 700 products for MSDOS, CP/M 86, CP/M 80, Macintosh and key products for other micros.

EDITORS Programming

	RUNS ON	OUR PRICE
BRIEF - Intuitive, flexible	PCDOS	195
C Screen with source	86/80	75
XTC - Multitasking	PCDOS	95
FINAL WORD-for manuals	86/80	215
MINCE-like EMACS	PC/80	149
PMATE-powerful	8086	185
VEDIT-full, liked	86/80	119

UNIX PC

COHERENT - for "C" users	PClike	475
COHERENT-NCI-Realtime	PClike	call
XENIX - plus C to MSDOS	PC	1275

Ask about run-times, applications, DOS compatibility, other alternatives. UNIX is a trademark of Bell Labs

LANGUAGE LIBRARIES

GRAPHICS: GraphiC-source in C	MSDOS	250
GRAPHMATIC-3D. FTN, PAS	PCDOS	125
HALO-fast, full-all lang.	PCDOS	145
FILE MGNT: BTrieve-all lang.	MSDOS	215
CIndex + -source, no royal.	86/80	375
CTree-source, no royal.	ALL	375
dBC ISAM by Lattice	8086	235
dB VISTA-"Network" Structure	MSDOS	465
PHACT-up under UNIX, addons	MSDOS	225
OTHER: CUtil by Essential	MSDOS	139
Greenleaf - 200 +	MSDOS	149
CSharp - **Real-Time**	MSDOS	600
PORTABLE C to PC, Mac, II	Many	125
SOFT Horizons - Blocks I	PCDOS	139
SCREEN: CURSES by Lattice	PCDOS	125
CView - input, validate	PCDOS	195
MetaWINDOW - icons, clip	PCDOS	139
PANEL - many lang, term	MSDOS	265
ProScreen - windows, source	PCDOS	415
Windows for C	MSDOS	175

FORTRAN

	RUNS ON	OUR PRICE
MS FORTRAN-86 - Impr.	MSDOS	$ 239
DR Fortran-86 - full '77'	8086	249
PolyFORTRAN-XREF, Xtract	PCDOS	165

OTHER PRODUCTS

Assembler & Tools - DRI	8086	159
Atron Debugger for Lattice	PCDOS	395
cEnglish - dBase to C	MSDOS	750
C Helper: DIFF, xref, more	86/80	135
CODESMITH-86 - debug	PCDOS	139
MacASM-full, fast, tools	MAC	115
MBP Cobol-86 - fast	8086	680
METAWINDOW - graph, fonts, clip	PCDOS	135
Micro: SubMATH-FORTRAN full	86/80	250
Microsoft MASM-86	MSDOS	125
MSD Debugger	PCDOS	119
Multilink - Multitasking	PCDOS	265
PC/FORTH + -well liked	MSDOS	219
PFIX-86 Debugger	MSDOS	169
PL/1-86	8086	495
Polylibrarian - thorough	MSDOS	95
PolyMAKE	PCDOS	95
PROFILER - flexible	MSDOS	125
Prolog-86-Learn, Experiment	MSDOS	125
SLK/F - Copy Protection	PCDOS	145
SYMD debugger-symbols	PCDOS	119
TRACE86 debugger ASM	MSDOS	115

Note: All prices subject to change without notice. Mention this ad. Some prices are specials. Ask about COD and **POs**. All formats available.

Call for a catalog, literature, and *solid value*

800-421-8006

THE PROGRAMMER'S SHOP™

128-D Rockland Street, Hanover, MA 02339

Visa Mass: 800-442-8070 or 617-826-7531 MasterCard 8517

Circle **no. 67** on reader service card.

fingered thing. I don't want to do nine kinds of multiple-key gymnastics, I don't want to move my hand from the home row of keys to a cursor pad and I certainly don't want to both move my hand from the home row *and* hold down multiple keys.

The cursor-control diamond in WordStar is reasonably comfortable for us touch typists; I can Control-F and Control-A to move forward and backward a word at a time almost as fast as I can type, but then I've had several books worth of practice. But WordStar's cursor controls are inadequate for moving around quickly in a large document. Here's a piece of user data: I find that I typically move screen page by screen page, then narrow in on my prey using word-by-word moves, with maybe a few up and down arrows. With a word processor that makes good use of a mouse, I can effectively narrow in on a particular spot on a screen, but it's less useful in larger moves, and there's still the issue of removing my hands from the home row.

All of which leads up to my notion of exponential cursor control. I only want six cursor control keys: four to move one character at a time, up, down, right and left; and two additional keys, a "power forward" and a "power backward" key.

My definition of a "power" key is one that moves the cursor in a fixed direction through the file, but that increases its step size with repeated presses (or if you hold the key down long enough for autorepeat to start). There are several algorithms possible for a power key:

Algorithm 1: time-dependent

```
WHILE power key held down DO
    WHILE time down < 1 unit DO
        move cursor one word
    WEND
    WHILE 1 unit < time down <
    2 units DO
        move cursor one sentence
    WEND
    WHILE 2 units < time down <
    3 units DO
        move cursor one paragraph
    WEND
    WHILE 3 units < time down <
    4 units DO
        move cursor one page
    WEND
WEND
```

Algorithm 2: context-dependent

```
flag=1
WHILE previous keypress was
power key DO
    CASE OF flag =
        1: move one word
           test for boundary
        2: move one phrase
           test for boundary
        etc.
    ENDCASE
WEND
```

The test for boundary sets the value of flag dependent on whether or not we've moved across some arbitrarily defined boundaries during the movement of the cursor:

phrase boundary: , .) ? ! ; :
sentence boundary: . ? !
paragraph boundary: <cr><lf>
page boundary: 66 lines
section boundary: section marker

With either approach, the effect is the same: the longer the power key is repeated, the greater distance it moves and the greater is its speed of movement, although the first approach requires that the user *hold* the power key down. Each approach has its advantages. The first allows you to lift your finger from the key momentarily to get your bearings and then restart at a slower cursor rate when you press the power key again. The second favors key tappers who use the rhythm of the keystrokes to reinforce what they're doing. Both approaches require programs that give priority to redisplaying the cursor and screen contents, since power keys would make the program especially sensitive to the "understeer" and "oversteer" problems that result from the display not keeping up with the cursor movement.

I've described moves in terms of units that most matter to me these days: words, sentences, paragraphs. Program editors, though, could be configured to use boundaries that make sense for the language being used. Boundaries could be set at keywords, statement boundaries, loop structures and functions, for example, if one were editing C programs. Better yet, give me an editor that lets me provide definitions for boundaries.

One last request to word-processing programmers: please keep my fingers on the keys. Use the WordStar diamond and let Control-A and Control-F be the power keys; I'll love it. I won't even object if you throw in screen-up and screen-down keys.

Micro Subscribers Please Note:

The Software Designer is a column devoted to the notion that the activity of programming engages one in a struggle with metaphors; that the only worthy goal in the struggle is to be a wielder of metaphors, a creator of metaphors, and not an unwitting tool of unexamined metaphors. That's what, I claim, distinguishes a software designer from a mere programmer: the software designer creates new metaphors for the machine.

A computer is, after all, a general purpose device, and that's a pretty ethereal concept. What does a general purpose device do? Well, it . . .

You see the point. Just to tell our friends what we are doing with what used to be our spare time, or what we do for a living, we have to create metaphors. All the metaphors are in fact just descriptions of software products: the Mac's metaphor is the desktop and VisiCalc gave us the spreadsheet metaphor. And when the metaphor is implemented, the computer, the general purpose device, acquires a specific purpose, becomes, briefly, an electronic desktop or spreadsheet. It's trivial to point out that the computer can do nothing until some programmer tells it what to do. But the computer *is* nothing until the software designer brings into the world a new metaphor, and with it a new intellectual tool.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 197.**

NEW! RELOCATABLE Z-80 MACRO ASSEMBLER
FROM MITEK

It's a real bargain! Here's why:

- Only $49.95 plus shipping
- 8080 to Z-80 Source Code Converter
- Generates Microsoft compatible REL files or INTEL compatible hex files
- Compatible with Digital Research macro assemblers MAC & RMAC
- Generates Digital Research compatible SYM files
- Full Zilog mnemonics
- INCLUDE and MACLIB files
- Conditional assembly
- Separate data, program, common and absolute program spaces
- Customize the Macro Assembler to your requirements with installation program
- Cross-reference Generation
- Z-80 Linker and Library Manager for Microsoft compatible REL files available as a total package with Macro Assembler for only $95.00 plus shipping
- Manual only is $15

TO ORDER, CALL TOLL FREE: 1-800-367-5134, ext. 804
For information or technical assistance: 1-808-623-6361

Specify desired 5¼" or 8" soft-sectored format. Personal check, cashier's check, money order, VISA, MC, or COD welcomed. Include $5 for postage and handling.

MITEK P.O. Box 2151
Honolulu, HI 96805

Z-80 is a trademark of Zilog, Inc. MAC, RMAC, and ZSID are trademarks of Digital Research, Inc.

Circle **no. 66** on reader service card.

MS-DOS, UNIX, Apple MAC, CP/M, NETWORKS and more. One c-tree ISAM DOES THEM ALL!

The company that introduced micros to B+ Trees in 1979 and created ACCESS MANAGER™ for Digital Research, now redefines the market for high performance, B+ Tree based file handlers. With c-tree™ you get:

- complete C source code written to K&R standards of portability
- high level, multi-key ISAM routines and low level B+ Tree functions
- routines that work with single-user and network systems
- no royalties on application programs

$395 COMPLETE

Specify format:
5¼" PC-DOS 3½" Mac
8" CP/M® 8" RT-11

for VISA, MC or COD orders, call
1-314-445-6833

c-tree™
BY FAIRCOM

2606 Johnson Drive
Columbia MO 65203

Access Manager and CP/M are trademarks of Digital Research, Inc. MS is a trademark of Microsoft c-tree and the circular disc logo are trademarks of FairCom UNIX is a trademark of Bell Laboratories Apple is a trademark of Apple Computer, Inc.

©1984 FairCom

Circle **no. 37** on reader service card.

Pascal and C Programmers

Your programs can now compile the *FirsTime*™

FirsTime is an intelligent editor that knows the rules of the language being programmed. It checks your statements as you enter them, and if it spots a mistake, it identifies it. *FirsTime* then positions the cursor over the error so you can correct it easily. *FirsTime will identify all syntax errors, undefined variables, and even statements with mismatched variable types.* In fact, any program developed with the *FirsTime* editor will compile on the first try.

More than a syntax checker!
FirsTime has many unique features found in no other editor. These powerful capabilities include a zoom command that allows you to examine the structure of your program, automatic program formatting, and block transforms.

If you wish, you can work even faster by automatically generating program structures with a single key-stroke. This feature is especially useful to those learning a new language, or to those who often switch between different languages.

Other Features: Full screen editing, horizontal scrolling, function key menus, help screens, inserts, deletes, appends, searches, and global replacing.

Programmers enjoy using *FirsTime*. It allows them to concentrate on program logic without having to worry about coding details. Debugging is reduced dramatically, and deadlines are more easily met.

FirsTime for PASCAL	$245
FirsTime for C	$295
Microsoft PASCAL Compiler	$245
Microsoft C Compiler	$395
Demonstration disk	$25

Get an extra **$100 off** the compiler when it is purchased with ***FirsTime.***
(N.J. residents please add 6% sales tax.)

Spruce
Technology Corporation
110 Whispering Pines Drive
Lincroft, N.J. 07738
(201) 741-8188 or (201) 663-0063

Dealer enquiries welcome. Custom versions for computer manufacturers and language developers are available.

FirstTime is a trademark of Spruce Technology Corporation.

MasterCard VISA

Circle **no. 65** on reader service card.

by Allen Holub

The topic of this month's column is sorting routines. We'll look at how the sort routine given in Kernighan & Ritchie works and talk about why you shouldn't use it. Then we'll present a general purpose Quicksort routine, modeled after the Unix routine qsort( ).

Shell Sort

Many C programmers, when they need a sort routine, grab the one out of *The C Programming Language* (p. 116). The routine works, it's short, why not use it? There are several reasons: it's not documented, it's slow, and better routines are available. Some of the slowness comes from using array indexes rather than pointers (an array index usually causes a multiply, while incrementing a pointer is an addition).

Unfortunately, the speed issue runs deeper. Just about every programming problem presents you with trade-offs, and sorting routines are worse than most in this respect. Not only are the usual considerations of code size versus efficiency a factor, but there are several possible ways to go about sorting an array, all of which behave differently. For example, with Quicksort, the time required to sort an array can change radically with the kind of data the routine is presented. You have to know how a sorting routine works before you can make an intelligent decision on whether to use it in a particular application.

The routine in K&R is a Shell sort. I thought for years that Shell sort was named after a shell game; anyone who has sat down with the K&R code and tried to figure out what's going on can understand how I came to this conclusion. As it turns out, the algorithm is named after its inventor, Donald Shell, who developed it in 1959. A Shell sort works by breaking up the sorting problem into a series of smaller problems. For example, a set of eight objects to be sorted is broken up into four subsets of two objects, each of which is sorted separately. The eight objects are then reorganized as two sets of four objects; again each is sorted separately. Finally, the set is sorted as one set of eight objects.

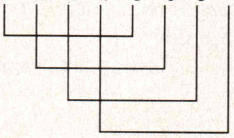
The rationale behind the Shell sort is that you get the more out-of-order parts of the original set into order very quickly. The array then is easier to sort on the next pass. What characterizes a Shell sort is the reorganization into smaller sets. Although the algorithm used to actually sort the items in each subset is immaterial, the behavior of this algorithm should get better as the set it is working on gets closer to being in order. A bubble sort does this. In fact, you can look at Shell sort as an improved bubble sort.

A concrete example may clarify the process. Let's start with a set of eight objects:

{9, 2, 1, 7, 3, 8, 5, 4}

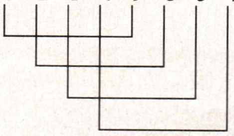
We break this up into four subsets consisting of the pairs {9,3}, {2,8}, {1,5}, and {7,4}:

9 2 1 7 3 8 5 4



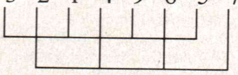
Then we sort each pair separately:

3 2 1 4 9 8 5 7



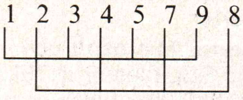
Next we break up the set into two subsets of four elements, {3, 1, 9, 5} and {2, 4, 8, 7}:

3 2 1 4 9 8 5 7



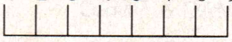
We then sort these separately:

1 2 3 4 5 7 9 8



Finally, we treat the input as one set of eight elements and just sort it:

1 2 3 4 5 7 8 9



Now, let's look at the routine in K&R (a slightly modified version appears in Listing One, page 94). The outermost for loop (line 6) determines how many elements will be in each subset; "gap" is the distance in the array between two elements of the same subset. In the above example, the distance between 9 and 3 (which compose the first subset) is 4; this would be the initial value of gap. Each time through this first for loop, we divide gap by two, which effectively doubles the number of elements in a subset.

The second and third for loops (line 7 and 8) are the actual sort routine. The algorithm is essentially a bubble sort, although the situtation is complicated by sorting all of the subsets at the same time. You can best see how the sort works by reducing the algorithm to handle only the last case (i.e., gap = = 1) and reversing the sense of the compare. This is done in Figure 1 (page 91).

In the innermost loop, the bubble sort scans along the unsorted array looking for two adjacent elements that are out of order. When it finds these elements, it exchanges them. In this way, an out-of-order element tends to "bubble" up to its correct place in the array. However, you have to do many comparisons and exchanges to get it there (worst case is an array already sorted but in reverse order; you have to make N–1 exchanges to get an out-of-place element from the right end of an N element array all the way to the left end). This inner loop must be executed enough times to guarantee that

all out-of-order elements find their way to the proper position in the array; therefore, a bubble sort will *always* use $O(n^2)$ comparisons. In the worst case, it will need almost as many exchanges. No wonder the routine is slow.

You can now see how the Shell sort strategy helps the bubble sort. It takes that pathologically out-of-place element and moves it to the front of the array in O(log N) exchanges, rather than in N–1 exchanges. You can improve the behavior of a Shell sort by fiddling with the increments between elements. This lets the various passes interact with each other in a more productive fashion. The increments should not be even multiples of each other; powers of two are actually among the worst increments. Knuth has determined that a good choice of increments is 1, 4, 13, 40, 121 .... He also says that the sequence 1, 3, 7, 15, 31, ... works well. Using this latter sequence of increments, a Shell sort will sort an N element array in $O(N^{1.2})$ time. This is better than the $O(N^2)$ time required for a bubble sort but not much. If you're interested, the gory details of this analysis are in Knuth's book *The Art of Computer Programming*, vol. 3 (Addison Wesley, 1973) pages 84f.

Quicksort

In the general case, a faster sorting routine than the Shell sort is Quicksort, developed by C. A. R. Hoare in 1962.

Quicksort, like the Shell sort, sorts an array in place; that is, it sorts by exchanging elements of an array rather than by copying. It works by selecting an arbitrary element of the array to be sorted (called the "pivot") and moving that element to its proper position in the sorted array. As it is doing this, it arranges the array so that all elements to the left of the pivot are less than the pivot and all the elements to the right are greater than the pivot. It then repeats this process recursively on the two halves of the array. It works as follows:

(1) Select an arbitrary element of the array to be sorted (the pivot). It's easiest just to lop off the last or first element of the array and use that as the pivot (we will use the last element in this example). However, if the pivot happens to be the largest or smallest element of the array, then the behavior of Quicksort degrades dramatically (Quicksort becomes Slowsort). You can avoid this worst case scenario by taking the center element as the pivot or, even better, by taking the median of three elements. This extra work will slow down the routine, but it's worth it if you expect to be sorting already sorted files.

(2) We will sort using two pointers called HIGH and LOW. To start sorting, make HIGH point at the next to

```
for( i = 1 ; i < argc ; i + + )
        for( j = i - 1; j > = 0 ; - -j )
                if( strcmp( argv[j], argv[j + 1] > 0)
                        exch( agv[j], argv[j + 1] )
```

Figure 1
Bubble Sort

	Initial LOW ↓								Initial HIGH ↓			Pivot ↓
Initial file:	**80**	20	40	100	110	30	10	0	160	120	**130**	90
Looking:	80	**20**	40	100	110	30	10	0	160	120	**130**	90
	80	20	**40**	100	110	30	10	0	160	120	**130**	90
	80	20	40	**100**	110	30	10	0	160	120	**130**	90
	80	20	40	**100**	110	30	10	0	160	**120**	130	90
	80	20	40	**100**	110	30	10	0	**160**	120	130	90
	80	20	40	**100**	110	30	10	**0**	160	120	130	90
1st exchange:	80	20	40	**0**	110	30	10	**100**	160	120	130	90
	80	20	40	0	**110**	30	10	**100**	160	120	130	90
	80	20	40	0	**110**	30	**10**	100	160	120	130	90
2nd exchange:	80	20	40	0	**10**	30	**110**	100	160	120	130	90
	80	20	40	0	10	**30**	**110**	100	160	120	130	90
Pointers cross:	80	20	40	0	10	**30**	**110**	100	160	120	130	90
				HIGH		↑	↑	LOW				
Exchange pivot:	80	20	40	0	10	**30**	**90**	100	160	120	130	110
Partitioned:	[80	20	40	0	10	30]	90	[100	160	120	130	110]
	↑ LOW				↑ HIGH	↑ Pivot		↑ LOW			↑ HIGH	↑ Pivot

Figure 2
Quicksort

```
Given:
        typedef struct
        {
                int     key, element1, element2, etc;
        }
        ELEMENT;

        compare( e1, e2 )
        ELEMENT *e1, *e2;
        {
                return( e1->key - e2->key );
        }
        ELEMENT array[10];
The array can be sorted with:
        qsort( array, 10, sizeof(ELEMENT), compare );
```

Figure 3
Sorting an Array of Structures

last element of the array and LOW point to the first element (the last element is the pivot).

(3) Move LOW towards the last element of the array. Stop when HIGH and LOW cross or when the object pointed to by LOW is *greater than* the pivot.

(4) Move HIGH toward the first element of the array. Stop when HIGH and LOW cross or when the object pointed to by HIGH is *less than* the pivot.

(5) If the two pointers have not crossed, exchange the objects pointed to by HIGH and LOW.

(6) Repeat steps 3, 4, and 5 until the pointers cross.

(7) Exchange the pivot and the object pointed to by LOW. This puts the pivot into its correct place in the array.

(8) The array is now partitioned so that all elements to the left of the pivot are less than the pivot and all elements to the right of the pivot are greater than the pivot. Repeat steps 1–8 on these two partitions (not including the pivot in either partition) until the array size of all arrays is less than or equal to one.

An example of the sorting process is given in Figure 2 (page 91). The objects pointed at by HIGH and LOW are underlined. For more information see *The Art of Computer Programming*, pages 114f.

The biggest problem with Quicksort is its sensitivity to the input data. Best case performance (when you always select the median as the pivot) requires O(N log N) comparisons and O(1/6 log N) exchanges. This is quite a bit better than O(N^2). The average case performance is still pretty good: O(2 ln 2) time slower than the best case. Worst case performance (when the pivot is the largest element in the array) is another matter. In this case, one partition is empty and the other contains the balance of the array (N–1). N (rather than log N) partitioning steps must be made, and the algorithm requires O(N^2) time to sort the array. This time is no worse than that of the bubble sort, but it's no better either.

A general purpose Quicksort routine is presented in Listing Two (page 94). This routine, called qsort( ), works like the Unix routine of the same name. By "general purpose" I mean that you can use qsort( ) to sort arrays of *anything*. You can sort arrays of pointers, of integers, of structures—of anything. The advantages of this versatility are obvious. The disadvantage is that qsort( ) will run slower than a routine tailored to a particular application.

There are two other potential problems with this particular implementation of Quicksort: it chooses its pivot in an unintelligent way and it's recursive. If you are likely to be sorting already sorted lists, you should modify the pivot selection to take a median value rather than just grabbing the last element. Handling the recursion is trickier. I've tried to minimize the average case recursion nesting, but the worst case is still pretty bad.

In particular, if you are sorting an already sorted array of N elements, there will be N–2 levels of recursion. Each level is passed two pointer-sized local variables on the stack. Assuming a 16-bit pointer, this yields eight bytes per level. The compiler will probably need a few other bytes for the return address and frame pointer, say, 12 bytes total (this is probably an underestimate). Sorting a 1024-element array would require about 12K of stack, not even counting the space needed for the array itself. The moral is to use this routine on small arrays. It's fast but it eats memory. Best case recursion nesting will require $\log_2 N$ levels, which is better but still not good. If someone has an iterative version of Quicksort, send it to me.

The versatility of qsort( ) comes from its not making any assumptions about the object being sorted. Its calling syntax is:

```
qsort( base, nel, width, compare )
char *base;
int nel, width;
int (*compare) ( )
```

"Base" is the base address of the array being sorted; it's declared as a character pointer but actually can be a pointer to anything. "Nel" is the size of the array in elements (as compared to bytes). "Width" is the size of one element in bytes, and "com-

pare" is a pointer to a comparison routine. This comparison should behave like strcmp( ); that is, when called with

(*compare) (a, b);

where a and b are *pointers* to two elements of the array, the comparison routine will return a negative number if $a < b$, zero if $a = = b$, and a positive number if $a > b$.

The main( ) routine in Listing Two (line 167) shows qsort being used to sort an array of character pointers. Figure 3 (page 91) shows it being used to sort an array of structures where the key field contains an integer sort key.

Qsort( ) itself (Listing Two, line 40) is an access routine. The actual sorting is done by rqsort( ) (line 74), which is static to the module (i.e., it isn't externally accessible). Qsort( ) first copies the two parameters that won't be modified (width and compare) into two global variables (line 63; comp and width are declared on line 20). Passing them as parameters to rqsort( ) would just increase the stack usage and make the subroutine calling overhead higher. Both variables are static, so you don't have to worry about their names conflicting with other global variables in your program. I have deviated a little from the Unix qsort( ) by defining a default comparison routine for sorting argv-like arrays of character pointers. To use the default routine, set the compare argument to 0. The actual sorting is started by calling rqsort( ) (on line 66).

Rqsort( ) is passed two parameters, "low" and "high" which point at the bottom and top elements of the array to be sorted. They have the same function as LOW and HIGH in the algorithm description. The local variable "pivot" is a pointer to the pivot (selected on line 91). Note that all the pointers are defined as character pointers. Because characters are of size 1, pointer arithmetic on a character pointer is plain arithmetic. We need to defeat pointer arithmetic because we don't know the size of the object at compile time. Adding or subtracting width from the pointer will advance it the correct number of bytes. The comparison routine has to know what's being pointed to, but at this level we need only the size.

The two while loops on lines 95 and 97 are looking for out-of-place elements, per steps 3 and 4 of the algorithm. Step 5, exchanging LOW and HIGH, is done on lines 99-110. Again, because we don't know the object size in advance, we have to exchange the two objects one byte at a time. We look for the pointers crossing at line 113. Step 7, the pivot exchange, is done on lines 118-124. Because an exchange is an expensive operation, "Low" and the pivot are exchanged only if necessary. On the other hand, the comparison itself takes time too, so you may want to pull it out of the code.

The partitioning is done on line 131. "Low" is advanced to exclude the pivot from both partitions (it's already in the correct place in the sorted array). Finally, step 8 (sorting the two partitions) is done on lines 131-145. Several tests are made here. The

test on line 132 determines which partition is smaller; the smaller partition is sorted first. This practice tends to reduce the number of recursion levels. The tests on lines 134, 136, 141, and 143 are looking for arrays with one or fewer elements in them; these arrays are already sorted. Most Quicksort algorithms put this test at the top of the algorithm, but this adds an additional level of recursion, and testing at the bottom doesn't add all that much extra code.

Conclusion

So, there are two sorting routines. Which you should use depends on the application. If you expect to sort a lot of already sorted data, the Shell sort will work in $O(N^{1.2})$ time while Quicksort needs $O(N^2)$. A Shell sort is also not the memory hog that Quicksort is. On the other hand, Quicksort is much faster on randomly sorted input, even though it eats stack space.

The qsort( ) routine given here is more versatile than the shell sort( ) routine, but it shouldn't be too hard to modify the latter to be as general purpose as qsort( ). Also, it's not hard to strip the versatility out of qsort( ) if you don't need it, improving its performance for a specific application.

I'd like to end this month with a request. If you have working C implementations of other sorting algorithms, send them to me. Perhaps we can do a follow-up of this column at some future date.

The O(N) Notation

The big O notation is often found when analyzing algorithm efficiency. It is read as "order of . . . ." O(N) is "order of N," $O(N^2)$ is "order of N squared, " and so on. "O(X) time" means that the algorithm will perform in time proportional to X. So, an algorithm that sorts an array in $O(N^2)$ time will sort the input data in time proportional to the number of input elements squared. To derive O(X), you look at the algorithm and extract the crucial operations, the ones that are performed most often. In the case of sorting algorithms, this will be some combination of comparison and exchange operations. You then figure out how many times that operation will be performed on a given set of input data. Because the main use of the big O notation is to compare algorithms, it's not particularly exact. You're just trying to express how an algorithm performs in a general sort of way. So, if an algorithm really takes $17 + N^2$ operations to do something, the 17 is ignored because it isn't particularly significant. In another example, Quicksort requires N*log N comparisons and 1/6(log N) exchanges in the general case. The 1/6 is relatively insignificant, so we usually say that Quicksort will work in O(N*log N) time in the general case.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 198.**

C Chest (Text begins on page 90)

Listing One

```
shell( argv, argc )
char     *argv[];
int      argc;
{
         int gap, i, j;

         for( gap = argc/2; gap > 0 ; gap /= 2)
                 for( i = gap; i < argc; i++ )
                         for( j = i-gap; j >= 0 ; j -= gap )
                         {
                                 if( strcmp(argv[j], argv[j+gap]) <= 0 )
                                         break;
                                 exch( &argv[j], &argv[j+gap] );
                         }
}
```

End Listing One

Listing Two

```
 1:  #include <stdio.h>

 2:  /*                            QSORT.C
 3:   *
 4:   *              Copyright (c) 1984 Allen I. Holub
 5:   *                    All rights reserved.
 6:   *
 7:   *      This program may be coppied for personal, non-profit use only.
 8:   *
 9:   *      Including a #define for DEBUG will make this file a stand-alone
10:   *      program which sorts argv and prints the result, along with all
11:   *      intermediate stages.  This is pretty instructive if you want to
12:   *      see how the quick sort works.
13:   */

14:  #ifdef DEBUG
```

(Continued on page 96)

Listing Two

```
static int        Lev = 0 ;         /* Recursion level */
static int        Maxlev = 0;       /* Maximum "       */
#endif

typedef int   (* PFI)();            /* pointer to a function returning int   */
static PFI    Comp ;                /* Pointer to comparison routine         */
static int    Width;                /* Width of an object in bytes           */

/*-----------------------------------------------------------------------*/

int     argvcmp( s1p, s2p )
char    **s1p, **s2p;
{
        /* Comparison routine for sorting an argv like list of pointers
         * to strings. Just remove one level of indirection and call
         * strcmp to do the comparison.
         */

#ifdef DEBUG

        register int    rval;
        rval =  strcmp(*s1p, *s2p) ;
        printf("level %d: argvcmp(<%s>,<%s>) = %d\n", Lev,*s1p,*s2p,rval);
        return( rval );
#else
        return( strcmp(*s1p, *s2p) );
#endif
}
qsort( base,  nel, width, compare )
char    *base;
int     nel, width;
int     (*compare)();
{
        /*      Perform a quick sort on an array starting at base. The
         *      array is nel elements large and width is the size of a
         *      single element in bytes. Compare is a pointer to a
         *      comparison routine which will be passed pointers to two
         *      elements of the array. It should return a negative number
         *      if the left-most argument is less than the rightmost,
         *      0 if the two arguments are equal, a positive number if
         *      the left argument is greater than the right. If compare
         *      is 0 then the default comparison routine, argvcmp
         *      (which sorts an argv-like array of prointers to strings),
         *      is used.
         */

#ifdef DEBUG
        printf("Sorting %d element array of %d byte elements at 0x%x\n",
                                                nel, width, base );

        printf("Comparison routine at 0x%x. Unsorted list:\n", compare);
        ptext( nel, base );
#endif

        Width = width;
        Comp  = (compare == (PFI)0) ? &argvcmp : compare ;

        if( nel > 1 )
                rqsort( base, base + ((nel-1) * width) );

#ifdef DEBUG
        printf("\nSort complete, list is:\n");
        ptext( nel, base );
        printf("Maximum recursion level = %d\n", Maxlev );
#endif
}

/*-----------------------------------------------------------------------*/

static          rqsort( low, high )
register char   *low, *high;
{
        /*      Workhorse function called by the access routine, qsort().
         *      Not externally accessible.
         */

        char            *pivot, *base;
```

(Continued on page 98)

DE SMET C

8086/8088 Development Package

$109

FULL DEVELOPMENT PACKAGE

- Full K&R C Compiler
- Assembler, Linker & Librarian
- Full Screen Editor
- Execution Profiler
- Complete STDIO Library (>120 Func)

Automatic DOS 1.X/2.X SUPPORT

BOTH 8087 & S/W FLOATING POINT

OVERLAYS

OUTSTANDING PERFORMANCE

- First and Second in AUG '83 BYTE benchmarks

SYMBOLIC DEBUGGER $50

- Examine & change variables by name using C expressions
- Flip between debug and display screen
- Display C source during execution
- Set multiple breakpoints by function or line number

DOS LINK SUPPORT $35

- Converts DeSmet.O to DOS.OBJ Format
- LINKs with DOS ASM
- Uses Lattice® naming conventions

C WARE CORPORATION

P.O. Box C, Sunnyvale, CA 94087

(408) 720-9696

Street Address: 505 W. Olive, #767 (94086) Call for hrs.

All orders shipped UPS surface on IBM format disks. Shipping included in price. California residents add sales tax. Canada shipping add $5, elsewhere add $15. Checks must be on U.S. Bank and in U.S. Dollars. Call 9am–1pm to CHARGE by VISA/MC/AMEX.

Foreign Distributors: **AFRICA**, HI-TECH SVCS, Gaborone 4540 or Telex 2205BD LANGER • **ENGLAND:** MLH Tech, 0606-891146 • **JAPAN:** JSE 03-486-7151 • **SWEDEN:** ESCORT DATA 08-87 41 48 **or** THESEUS KONSULT 08-23 61 60

Circle no. **18** on reader service card.

Thanks to YOU . . . We're Growing . . .
with YOU and your Computer . . .

LEO ELECTRONICS, INC.
P.O. Box 11307
Torrance, CA 90510-1307
Tel: 213/212-6133 800/421-9565
TLX: 291 985 LEO UR

We Offer . . . PRICE . . . QUALITY . . . PERSONAL SERVICE

64K UPGRADE

9 Bank	(IBM PC)	$26.10	(150ns)
		$24.75	(200ns)
4164	(150ns)	$2.90 ea.	
	(200ns)	$2.75 ea.	
8 Bank	(other PC)	$23.20	(150ns)
		$22.00	(200ns)
4164	(150ns)	$2.90 ea.	
	(200ns)	$2.75 ea.	

256K "MOTHER-SAVER" UPGRADE

256K	— (150ns)	$22.00 ea.	
6116P-3	— $4.00	2732	— $3.80
2716	— $2.95	2764	— $5.50
TMS-2716	— $4.95	27128	— $16.50

We accept checks, Visa, Mastercard or Purchase Orders from qualified firms and institutions. U.S. funds only. Call for C.O.D. California residents add 6½% tax. Shipping is UPS. Add $2.00 for ground and $5.00 for air. All major manufacturers. All parts 100% guaranteed. Pricing subject to change without notice.

Circle no. **59** on reader service card.

6 TIMES FASTER!

SuperFast Software Development Tools

INCREASE YOUR PROGRAMMING EFFICIENCY
with high-performance software development products from SLR Systems.
No other tools approach the speed or flexibility of the SLR Systems line.

"Z80ASM is an extraordinary product...",
Robert Blum, Sept. 84 DDJ

"...in two words, I'd say speed & flexibility",
Edward Joyce, Nov. 84 Microcomputing

ASSEMBLERS

- RMAC/M80 macros
- Nested INCLUDES & conditionals
- 16 char. labels on externals
- Built in cross-reference
- Optional case significance
- Phase/dephase
- Math on external words and bytes
- Define symbols from console
- Generate COM, HEX, SLR-REL, or Microsoft-REL files
- Time & Date in listing
- Over 30 configure options

Z80ASM -full Zilog Z80 .... $125
NEW! Z80ASM+ -all tables virtual $195
NEW! SLRMAC -full Intel 8080, with Z80.LIB extensions internal $125
NEW! SLRMAC+ -all tables virtual $195

Z80 CPU, CP/M compatible, 32K TPA required.

"Z80ASM...a breath of fresh air...",
Computer Language, Feb. 85

C.O.D., Check or Money Order Accepted

LINKERS

- Links SLR & M80 format files
- Output HEX or COM file
- Three separate address spaces
- Load map and SID/ZSID .SYM file

SLRNK+ includes:

- All tables overflow to disk
- HEX files do not fill unused space
- Intermodule cross-reference
- EIGHT separate address spaces
- Works with FORTRAN & BASIC
- Generate PRL & SPR files
- Supports manual overlays
- Full 64K output

SLRNK -fastest memory based $125
NEW! SLRNK+ -full featured virtual $195
Combo Paks available from $199. - $299.

For additional information contact SLR Systems

1-800-833-3061, in PA (412) 282-0864
1622 N. Main St., Butler, PA 16001 • Telex 559215

SLR Systems

Circle no. **78** on reader service card.

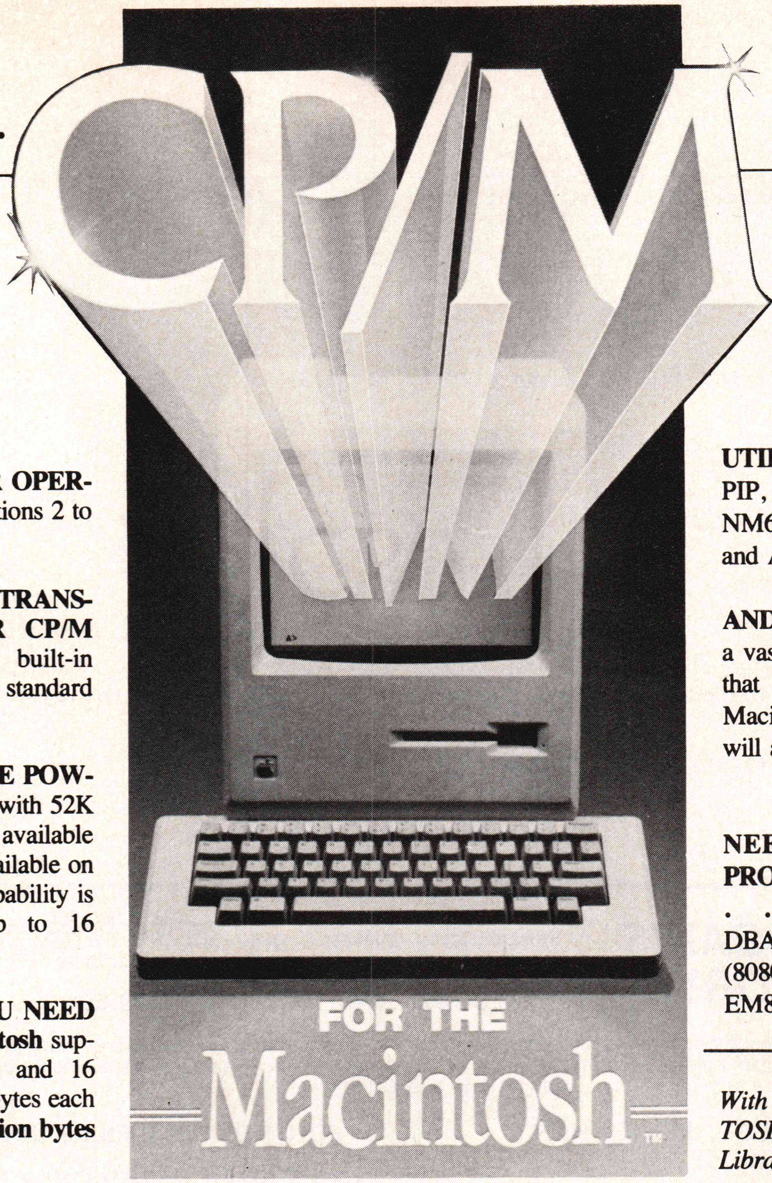
C Chest (Listing Continued, text begins on page 90)

Listing Two

```
        static char     *a, *b;           /* Used for exchanges, will not */
        static int      tmp, i;           /* need to retain their values  */
                                          /* during the recursion so they */
                                          /* can be static.               */
#ifdef DEBUG
        printf("New pass, recursion level %d\n", Lev );
        if( Lev > Maxlev )
                Maxlev = Lev;
#endif
        base = low    ;           /* Remember base address of array */
        pivot = high  ;           /* Partition off the pivot        */
        high -= Width ;

        do
        {
                while( low < high && (*Comp)(low, pivot)  <= 0 )
                        low += Width;

                while( low < high && (*Comp)(high, pivot) >= 0 )
                        high -= Width;

                if( low < high )              /* exchange  low  &  high  */
                {
#ifdef DEBUG
                        printf("lev %d: exchanging high:<%s> & low:<%s>\n",
                               Lev,*((char **)high), *((char**)low));
#endif
                        for( b = low, a = high, i = Width ; --i >= 0 ;)
                        {
                                tmp    = *b   ;    /* Exchange *low and *high */
                                *b++   = *a   ;
                                *a++   = tmp  ;
                        }
                }

        } while ( low < high );


#ifdef DEBUG
        printf("level %d: Exchanging pivot:<%s>  & low:<%s>\n",
                        Lev, *((char **)pivot), *((char**)low) );
#endif

        if( low < pivot  &&  (*Comp)(low, pivot) > 0 )
                for( b = low, a = pivot, i = Width ; --i >= 0 ;)
                {
                        tmp    = *b   ;      /* Exchange *low and *pivot */
                        *b++   = *a   ;
                        *a++   = tmp  ;
                }
#ifdef DEBUG
        printf("\nDone with pass, partially sorted list =\n");
        ptext( ((pivot - base)/Width) + 1 , base );
        printf("\n");
        ++Lev ;
#endif

        low += Width;

        if( high - base  <  pivot - low )
        {
                if( low < pivot )
                        rqsort( low, pivot );

                if( base < high )
                        rqsort( base, high );
        }
        else
        {
                if( base < high )
                        rqsort( base, high );

                if( low < pivot )
                        rqsort( low, pivot );
        }
```

(Continued on page 100)

The Macintosh Software Library just got fatter . . .



It's about TIME. AND SPEED.

FASTER COMPUTER OPERATION with disk operations 2 to 5 times faster.

QUICK PROGRAM TRANSFER FROM OTHER CP/M COMPUTERS using built-in transfer utilities and standard CP/M file structure.

RUNS BIGGER MORE POWERFUL PROGRAMS with 52K transient program area available on 128K Mac, 300K available on 512K system and the capability is built-in to manage up to 16 Megabytes of RAM.

PLUG-IN WHAT YOU NEED ...CP/M for the Macintosh supports up to 2 printers and 16 Drives of 512,000,000 bytes each for a maximum of **8 Billion bytes of storage.**

MORE DISK SPACE AVAILABLE ... each disk drive has 360 bytes available, often eliminating the need for additional drives.

CHOOSE ANY PRINTER letter quality or dot matrix ... any serial printer will work just fine.

MEDIA PROBLEMS?...NONE ... all that is needed to transfer programs from 5¼ to 3½ is a modem transfer program to operate with our Modem 7 Communications Compatible Program ... this enables programs to be transferred from **XEROX, OSBORNE, HEATHKIT, KAYPRO, RADIO SHACK** and others.

THE 10 FEATURES:

1. 6 Disk System
2. Documentation from Digital Research and I.Q.
3. MacroAssembler ***(no extra charge)***
4. Modem 7 Compatible Communications Transfer Program ***(no extra charge)***
5. C Compiler ***(no extra charge)***
6. Text Editor ***(no extra charge)***
7. Menu Program (puts menus on your programs) ***(no extra charge)***
8. Standard Printer Driver ***(no extra charge)***
9. Copy Program ***(no extra charge)***
10. Leir-Seigler ADM3A Terminal Emulation Program ***(no extra charge)***

All This For Only $395 Retail

UNLIMITED EXPANDABILITY is made possible with the total and easy control of serial expansion ports which allows many forms of plug-ins.

UTILITIES ... such as STAT, PIP, DDT, ED, LINK68, INIT, NM68, SIZE, LO68, RELOC, and AS68.

AND We are currently compiling a vast Library of CP/M software that can be ported over to the Macintosh and our Macrolibrary will also be available soon.

NEED POPULAR WORD PROCESSING PROGRAMS? . . . Run WORDSTAR® DBASEII® AND other CP/M (8080) 2.2 Programs with our EM80 Emulator . . **ONLY $195.!**

With CP/M FOR THE MACINTOSH the Macintosh Software Library just got fatter!

Call or write for your Macintosh CP/M Software Catalog.

IQ software

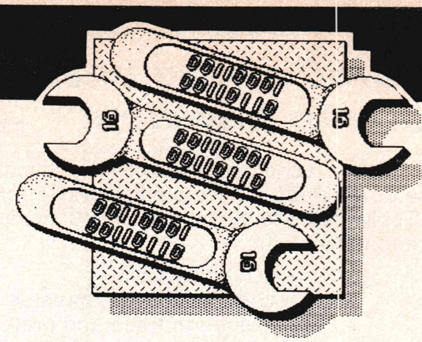
I.Q. Software
2229 E. Loop 820 N.
Ft. Worth, Texas 76118
(817) 589-2000

 (Listing Continued, text begins on page 90)

Listing Two

```
#ifdef DEBUG
        --Lev ;
#endif
}
/*----------------------------------------------------------------------*/
/*      Test routine for qsort. compiled if DEBUG is #defined          */
#ifdef DEBUG
static ptext( argc, argv )
int     argc;
char    **argv;
{
        /*      Print out argv, one element per line.
         */

        register int    i;

        for( i = 1; --argc >= 0 ; i++ )
                printf("%2d: %s\n", i, *argv++ );
}

main(argc, argv)
int     argc;
char    **argv;
{
        /* Test routine for qsort. Sorts argv (less the first element).
         */

        qsort( ++argv, --argc, sizeof(PFI) , 0 );
}
#endif
```

End Listings



by Ray Duncan

A New Assembler for MSDOS

Philip Oliver has sent us a copy of his new product ED/ASM-86 for review. This is an interactive programming tool that combines the functions of an assembler, line editor, linker, and debugger in a single 50K program. 8087, 80186 and 80286 mnemonics are fully supported for assembly, tracing, and disassembly. Macros, structures, and conditional assembly directives are included. This is an interesting product, although I feel that its appeal would be broader if it were fully source compatible with the Microsoft Macro Assembler and if it could handle standard Microsoft Linker object files. The current price of ED/ASM-86 is $100.00; contact Oliver Computing Company at 9311 Jutland Court #D, Indianapolis, IN 46250 for more information.

TEE Filter for MSDOS 2

Mr. A. K. Head, of Melbourne, Australia, contributed a pair of MSDOS 2.x filters. We are printing the first of these, called TEE, in this month's column as Listing One (page 106). TEE, a standard filter in Unix systems, is used in a pipe to make two copies of the standard input, directing one to a file and the other to the standard output (which might also be redirected into a file). Where you normally would use a filename, here you can also use any logical device such as PRN: or CON:. For example, you could enter the command:

```
DIR | SORT | TEE C:SORTED.DIR
```

This would direct the output of the directory command to the SORT filter and thence to TEE, which would send one copy of the sorted listing to the disk file C:SORTED.DIR and the other copy to the standard output (console, in this case). This version of TEE doesn't support the Unix switches –i (ignore interrupts) or –a (append to previous contents of a file).

80286 Feedback

Matt Robins, of Beverly Hills, California, writes: "I am writing to you concerning two and a half errors in your November 1984 *DDJ* column.

"PCDOS 3.0 could not provide the '. . . same old horrible EDLIN that we first met in PCDOS 1.0.' In fact, as you probably know by now, the PCDOS 2.0 version of EDLIN is a significant improvement on the PCDOS 1.1 version that I started out with. It boasts 40% more commands (14 vs. 10) of the most advanced variety: Copy-line and Move-line that make programming easy. Additional capability is provided by the use of + and – signs, like old CP/M ED (which *was* a true abomination).

"What I particularly like about the new EDLIN is that it uses the DOS edit mode keys F1 through F4. I couldn't live without this capability! I've enhanced my EDLIN by assigning the arrow keys and Home, End, PgUp, and PgDn keys to ANSI sequences that execute the appropriate commands in EDLIN (to comply with the above mentioned keytop definitions) . . . .

"Your second mistake was calling the 80286 'a slightly enhanced 8086.' By now you must have read the November *BYTE* article by Paul Wells that correctly states: '. . . at the same clock speed and [when you] do not take advantage of any 80286 capabilities, the 80286 achieves 250 percent of the 8086's performance.' It does that, partly, by having five processors on board, as opposed to the 8086's three."

I agree that the 80286 in protected mode is a very different machine, but I still maintain that you don't gain much more by using an 80286 in real mode (as PCDOS 3.0 does) than you could get just by cranking up the clock on a normal 8086 and using 16-bit memory like the Compaq DeskPro.

Matt goes on to say: "Finally, a half-a-mistake (maybe only a matter of taste) is your saying that use of the multitasking, virtual memory, and memory protection will '. . . wring all of the available computing power out of the 80286.' I tend to think of computing power as throughput, and the three activities mentioned above lower the throughput, not facilitate it. Put another way, 'computing power' is benchmark timings, whereas the three activities above are memory management functions. Being a member of the 'one [or more] processors, one user' school of thought, I am interested in Unix and multitasking only for my personal use: performing several of *my* tasks simultaneously."

More Macintosh Feedback

Steve Rabalais writes: "The Macintosh contains a rich operating system that, if used right, will take a lot of the labor out of programming. Utilities for Floating Point (IEEE Standard), File Management, Transcendentals, etc., are easily accessed from Assembly Language. The Apple people supporting the development work are apparently impressed by the BIG NAME programming shops, and independent developers are left to their own wits (which is probably a good thing). I personally do not depend on phone calls to solve programming problems and prefer to 'dig it out' because this is the best way to learn. I attempted to become an Apple Certified Developer but failed to impress the Marketing and Sales people with

their requisite pile of paperwork. I purchased my Macintosh and Lisa 'over the counter' and began programming immediately. I have talked to Certified Developers that had to wait months to obtain their 'favored' hardware discounts. I do have 'INSIDE MACINTOSH' and have found it indispensable for writing Mac applications.

"Although I do not have the official Macintosh Apple Assembler, I have heard horror stories to the effect that it requires two Macs. I use an excellent, inexpensive Assembler called *MacAsm* from Mainstay that allows me to program complete applications on a 128K Macintosh with a single 400K disk drive. MacAsm is a two pass Macro Assembler and is complete with a Resource Compiler and Text Editor. I am writing a series of Macintosh CADD (computer-aided design and drafting) applications and have had no problems using this assembler. I have found it fast, efficient, and simple. I am *not* using the Apple Workshop on the Lisa to develop software because it is slow and complicated to use and Apple recently wanted more money for a complete set of the latest updates.

"I use the Workshop version I have to help learn 68000 assembly language. The Workshop contains a Pascal compiler that will dump the generated assembly code into a text file. I inspect the code, figure out how things are done, and then rewrite it (for efficiency and clarity) using MacAsm. This may sound like cheating, but I find that I am gradually depending less on the Lisa as the learning process continues. Macintosh programming in assembly language is *hard* to learn, but once you have learned, it is *easy* to write sophisticated applications. Independent Developers have made the Apple II successful and apparently will do the same for the Macintosh in spite of the roadblocks. In conclusion, I would say that the Macintosh is a superb combination of hardware and software, and a proper programming environment can be had at a reasonable price . . . . Hopefully, other programmers will read this and know that there is help in the wilderness for programming the Macintosh."

Steve also contributed a programming tip for the 68000, which appears as Listing Two (page 110) following this column.

80286 XENIX

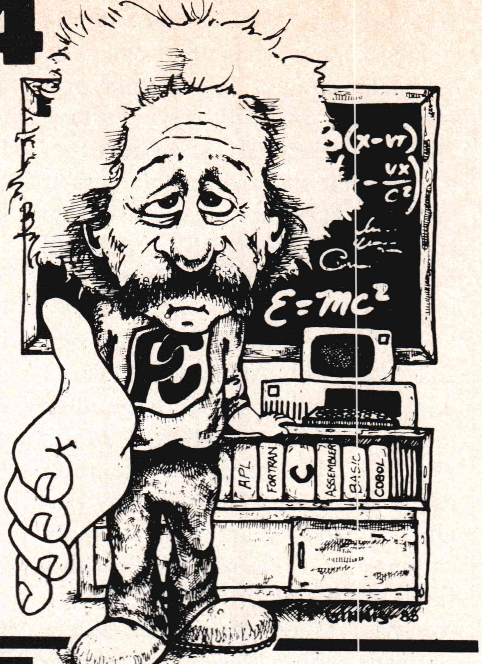
Speaking of Unix, we have been using 80286 XENIX 3.0 for the PC/AT here at Laboratory Microsystems for about two months. Getting acquainted with Unix is an entertaining but aggravating experience. Unix devotees claim that it was designed by programmers for programmers and increases productivity enormously, but to me it looks like it grew up topsy turvy out of the late-night efforts of a legion of graduate student hackers. No doubt, back when Ken Thompson and Dennis Ritchie invented Unix, it was compact and elegant (after all, it ran on a PDP-7), but it has grown into a behemoth that needs 256K of RAM to get off the ground.

The "Adventure" game mentality is seen everywhere in the system. "You are lost in a twisty, turning maze of cryptic, poorly indexed software manuals." There is no uniformity of syntax whatsoever, the commands have obscure or arcane names (grep, ls, cat), switches are case-sensitive (for instance, a lower-case "s" switch and an upper-case "S" switch on the C compiler do vastly different things), and the screen editor *vi* makes WordStar look positively user friendly.

The documentation for XENIX is clearly derived from the original Unix manuals, even though it has been filtered through the technical writers at Microsoft and then IBM. One often comes across unexpectedly witty sayings, such as (under the SPLINE command) "a limit of 1000 points is silently enforced" or (in the terminal capabilities library) "Hazeltime = brain damage" and "Names starting with X are reserved for serious disturbances." Command names frequently have a high cuteness factor, such as "finger" (report names of active users), "whodo," and "stty sane" (used when the operating system loses its mind and starts spitting garbage out to your terminal).

None of this is meant as a criticism

ADVANTAGE #4



At Programmer's Connection we listen to programmers and we take the time to research and test our products. We are confident in our knowledge of the software market and that's why we welcome your inquiries. Our goal is to help you make an informed decision when purchasing a programming language or utility. Call us today — you'll discover the difference. Programmer's Connection will revolutionize the way you think about software development.

Discover the advantages of buying from Programmer's Connection:

1. We offer the latest version of a product.
2. Most popular products are in stock ready to be shipped.
3. Receive same manufacturer's support as if buying direct.
4. Experienced professional programmers are on staff.
5. Choose from a large selection of the best software products available.
6. Knowledgeable and courteous sales staff.
7. Significant discounts off of retail prices.
8. No extra charge on prepaid orders, including major credit cards.
9. Reasonable charges for shipping and handling.
10. Toll free services from Canada and the Continental U.S.

Programmer's Development Tools:

C LANGUAGE:

	List	Ours
Computer Innovations C-86 Compiler	395	**299**
DeSmet C Compiler with Debugger	159	**145**
Lattice C Compiler from Lifeboat	500	**299**
Mac C by Consulair *for MacIntosh*	295	**259**
Mark Williams C Compiler w/Source Debugger	500	**449**
Xenix Development System by SCO	1350	**1099**

Special Combination Offer
Lattice C Compiler and
C-SPRITE Symbolic Level Debugger
Combined List Price $675 Our Price **$429**

OTHER LANGUAGES:

8088 Assembler w/Z-80 Translator 2500 AD	100	**89**
APL*Plus/PC by STSC	595	**499**
BetterBASIC by Summit Software	200	**169**
Golden Common LISP by Gold Hill	495	**439**
Macro Assembler by Microsoft . *New Release*	150	**119**
Modula-2/86 by Logitech	495	**439**
Professional BASIC by Morgan Computing	95	**89**

C UTILITIES:

Asynch Comm Library by Greenleaf	160	**129**
C Power Paks from Software Horizons	Call	**Call**
C-Sprite *Symbolic Debugger for Lattice*	175	**159**
C Utility Library by Essential Software	149	**119**
dBC *dBase/C Interface* by Lattice	250	**219**
DOS LINK Support *for DeSmet C*	35	**35**
English-to-C/C-to-English by Catalytix	100	**100**
ESP for C by Bellesoft	349	**279**
Graphic C by Scientific Endeavors	195	**169**
Greenleaf C Functions Library	175	**129**
Halo *Graphics* by Media Cybernetics	200	**125**
PANEL *Screen Editor* by Roundhill	295	**234**
Run/C Interpreter by Age of Reason	150	**129**

Introducing Pre-C by Phoenix Software
Complete lint-like utility that helps detect logic errors by searching for inconsistencies in functions and data types across multiple files.
List Price $395 Our Price **$339**

C UTILITIES:

Safe C Standalone Interpreter by Catalytix	400	**400**
Safe C Dynamic Profiler by Catalytix	150	**150**
Safe C Runtime Analyzer by Catalytix	400	**400**
Windows For C by Creative Solutions	195	**139**

c-tree by Faircom
Full featured B-Tree functions for high speed ISAM file management. Comes as C source code which can be compiled on almost any system including MacIntosh. *No royalties on generated code.*
List Price $395 Our Price **$359**

OTHER PRODUCTS:

APL2C by Decision Images *Interfaces APL to C*	150	**139**
Btrieve by SoftCraft	250	**199**
Dr. Halo by Media Cybernetics	95	**79**
FORTRAN Libraries by Alpha Comp. Serv.	Call	**Call**
FORTRAN Scientific Subroutine Library	175	**159**
Periscope *Debugger* by Data Base Decisions	295	**269**
Pfix-86 Plus by Phoenix	395	**299**
Plink-86 *Overlay Linker* by Phoenix	395	**299**
Pmate *Macro Text Editor* by Phoenix	225	**159**
Polytron Products ...... *We Carry a Full Line*	Call	**Call**
Profiler by DWB Associates	125	**89**
Screen Sculptor by Software Bottling	125	**109**
XTC *Text Editor* by Wendin	99	**89**
Xtrieve by SoftCraft .... *Sale!*	195	**149**

CODESMITH-86 Symbolic Debugger by Visual Age
New version 1.9 provides dual-mode patching assembler, branch-to-patch mode, stop-on-data compare/mis-compare, dual monitor debug mode, breakpoints and passpoints, machine state snapshot and hotline technical support.
List Price $145 Special Price **$119**

Prices are subject to change without notice.
Account is charged when order is shipped.

VISA MasterCard

Call for our new Spring Catalog

1-800-336-1166 In Canada: **1-800-225-1166**

Programmer's Connection
136 Sunnyside Street
Hartville, Ohio 44632
(216) 877-3781 (In Ohio)

"Programmers Serving Programmers"

Circle **no. 134** on reader service card.

of the Microsoft implementation, however, because the boys in Bellevue seem to have done a pretty good job of implementing System III on the 80286. And this was no weekend fling, either. IBM states that "the total number of lines that migrated from the original AT&T source code to PC XENIX was well over 400,000 lines."

The performance, even with a couple of extra terminals hung onto the system, is quite reasonable—due mainly to the PC/AT's very fast hard disk. XENIX 3.0 runs the 80286 in protected mode and is very crash-proof compared to PCDOS. Any attempt by your program to access memory not specifically allocated to your program, or to execute an illegal or privileged opcode, results in its immediate termination. We have found that as you develop new software under XENIX, the friendly message "Memory fault—core dump" will be your constant companion.

A few warnings before you rush out to buy a PC/AT and XENIX. There is no operating system support for graphics display modes; there is only very primitive support for the keyboard (ALT-key combinations can't be read, and even reading the unadorned function or arrow keys must be done through writing control strings to the ANSI driver); and the C compiler apparently has bugs in the "large model" and the optimizer, as Microsoft explicitly warned us not to use those features.

Because 80286 XENIX runs in protected mode, direct access to the hardware is, for all practical purposes, nonexistent—you can forget about your carefully optimized, memory-mapped video routines and all the other code you've painstakingly created to make your PCDOS applications look flashy. The ROM BIOS isn't available either. In fact, your program can't even tell where it is running, let alone where anything else is: the segment registers contain only logical selectors, and the corresponding physical addresses are not visible. Those of us who have become spoiled by the ability on micros to reach out and twiddle bits or ports wherever we please are out of luck.

One more warning, and not the least: the operating system requires some 15 Mbytes of hard disk space. If you are planning to use both XENIX and PCDOS on your PC/AT, best to plan ahead by buying *two* hard disks.

At the UNIFORUM conference in Dallas that we attended in January, Microsoft discussed plans to upgrade 80286 XENIX for compatibility with AT&T's Unix System V. Where this will leave purchasers of IBM PC/AT XENIX is not clear. The PC/AT version is being sold and supported directly by IBM, and, as we have found out to our sorrow with other products (such as the Microsoft Assembler), new improved versions released by Microsoft hardly ever seem to migrate down the pipeline to show up in IBM distribution.

But however you slice it, the release of PC/AT XENIX is a significant event for the Unix community. This is the first time that a personal computer powerful enough to run

Unix with decent performance has been available at a price the average small businessman would be willing to pay; the added prestige of the three magic initials behind it (no, I don't mean AT&T) may be sufficient to let it travel into uncharted regions where no Unix has gone before. The total installed base of Unix and Unix-derivative systems in the world is currently estimated to be less than 100,000; IBM could conceivably double or triple this in a year with the right kind of marketing and application software. It should be interesting to see what happens.

DDJ

16-Bit (Text begins on page 102)

Listing One

```
        name    tee
        page    55,132
        title   'TEE - tee junction for DOS pipes'
;
; TEE --- A "tee" junction for DOS 2.x pipes
;         after the fashion of the Unix function TEE
;
; By A. K. Head, 6 Duffryn Place, Melbourne, Australia 3142
;    reformatted and error handling added by Ray Duncan
;
; TEE reads from the standard input (redirectable) and
; outputs to both the standard output (redirectable)
; and a file, e.g.:   DIR | TEE C:\MYDIR\FILE.EXT | SORT
;
; The file can be CON, LPT1, etc.  If it is CON, then
; keyboard Control-S pauses the display as usual.

command equ     80h             ; buffer for command tail
buflen  equ     16384           ; buffer length, alter to taste

cr      equ     0dh             ; ASCII carriage return
lf      equ     0ah             ; ASCII line feed
ff      equ     0ch             ; ASCII form feed
eof     equ     01ah            ; End-of-file marker
tab     equ     09h             ; ASCII tab code
blank   equ     20h             ; ASCII blank

                                ; DOS 2.x pre-defined handles
stdin   equ     0000            ; standard input file
stdout  equ     0001            ; standard output file
stderr  equ     0002            ; standard error file
stdaux  equ     0003            ; standard auxilliary file
stdprn  equ     0004            ; standard printer file


cseg    segment para public 'CODE'

        assume  cs:cseg,ds:cseg

        org     100H            ; start .COM at 100H

tee     proc    far

        mov     cl,ds:command   ; get length of command tail.
        xor     ch,ch
                                ; address of command string.
        mov     si,offset command+1
        mov     di,offset command+1
tee1:                           ; squeeze out leading spaces.
        mov     al,byte ptr [si]
        cmp     al,blank
        jbe     tee2
        mov     byte ptr [di],al
        inc     di

tee2:   inc     si              ; look through command tail
        loop    tee1            ; until it's exhausted.
        mov     byte ptr [di],0 ; make ASCIIZ string.
```

(Continued on page 108)

DDJ Classifieds

Software

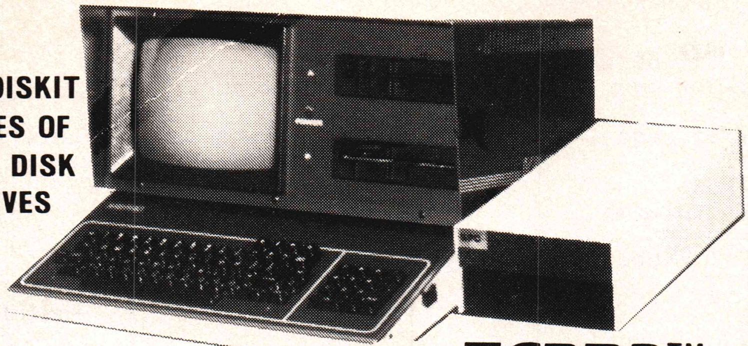
Listing One

```
        mov     ah,3ch          ; create output file.
        mov     cx,0            ; attribute=0.
        mov     dx,offset command+1
        int     21h
        jc      tee6            ; can't create file
        mov     handle,ax       ; save token for file.

tee3:   mov     ah,3fh          ; read standard input.
        mov     bx,stdin
        mov     cx,buflen
        lea     dx,buffer
        int     21h
        jc      tee4            ; jump, error.
        mov     nchar,ax        ; save length of read.
        cmp     ax,0            ; nothing read in?
        je      tee4            ; end of file, exit.
        mov     ah,40h          ; write to file...
        mov     bx,handle
        mov     cx,nchar
        int     21h
        jc      tee7            ; jump, write failed.
        cmp     ax,nchar
        jne     tee7            ; jump, write failed.
        mov     ah,40h          ; now write to standard output...
        mov     bx,stdout
        int     21h
        jc      tee7            ; jump, write failed.
        cmp     ax,nchar
        jne     tee7            ; jump, write failed.
        jmp     tee3            ; read again until end of file.

tee4:   mov     ah,3eh          ; close output file.
        mov     bx,handle
        int     21h

tee5:   mov     ax,4c00h        ; exit with return code=0.
        int     21h

tee6:   mov     dx,offset err1  ; print "Can't create file".
        mov     cx,err1len
        mov     ah,40h          ; display error message and exit.
        mov     bx,stderr
        int     21h
        mov     ax,4c01h        ; exit with return code=1.
        int     21h

tee7:   mov     dx,offset err2  ; print "Disk is full".
        mov     cx,err2len
        mov     ah,40h          ; display error message and exit.
        mov     bx,stderr
        int     21h
        mov     ah,3eh          ; close output file.
        mov     bx,handle
        int     21h
        mov     ax,4c02h        ; exit with return code=2.
        int     21h

tee     endp

nchar   dw      0               ; number of chars actually input.
handle  dw      0               ; token for output file.

err1    db      cr,lf
        db      'tee: Cannot create file'
        db      cr,lf
err1len equ     (this byte)-(offset err1)

err2    db      cr,lf
        db      'tee: Disk is full.'
```

(Continued on page 110)

FOR THE SERIOUS KAYPRO® USER

THE DISKIT SERIES OF HARD DISK DRIVES



...now with **ZCPR3™**

Now you can add from 5 to 40 Megabytes of fast-access Winchester storage to your KAYPRO 2, 4, or 10. The DISKIT is only 4 inches high; 5.7 if you get the two drive model with the *removable* 5 or 10 Mb. cartridge, and weighs less than 10 pounds. Easily disconnect DISKIT from the computer whenever you want, and if more capacity is required, just swap your drive for a larger model.

Our DISKIT Model 10 has 10.8 Megabytes of *formatted* capacity . . . **20%** more than a Kaypro 10, and runs about twice as fast. Installs in minutes.

Call SPC now and ask for more information. Quantity and prepayment discounts are available.

SYSTEMS PERIPHERALS CONSULTANTS
9747 Business Park Avenue
San Diego, CA 92131
(619) 693-8611

Circle **no. 104** on reader service card.

$5.00 C Compiler

Due to popular demand, **Dr. Dobb's Journal** has reprinted its most-asked-for C compiler articles by Ron Cain and J. E. Hendrix, each for only $5.00.

Ron Cain's C compiler from sold-out 1980 issues #45 and #48 includes "A Small C Compiler for the 8080s" and "Runtime Library for the Small C Compiler."

The J. E. Hendrix reprint includes part two of "Small-C Compiler v.2" from sold out issue #75 and completes the first part of the compiler article from issue #74 which is included in Dr. Dobb's Bound Volume 7.

To Order: Enclose $5.00 for each copy with this coupon and send to:
Dr. Dobb's Journal, 2464 Embarcadero Way, Palo Alto, CA 94303
Outside U.S., add $2.00 per copy for shipping and handling.

Please send _____ copy(ies) of the Ron Cain Reprint, and
_____ copy(ies) of the J. E. Hendrix reprint to:

Name ______________________________

Address ______________________________

City ____________________ State _____ Zip _______

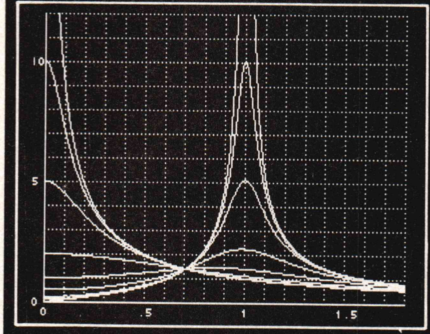
ALL REPRINT ORDERS MUST BE PREPAID.

Please allow 6–9 weeks for delivery. 102

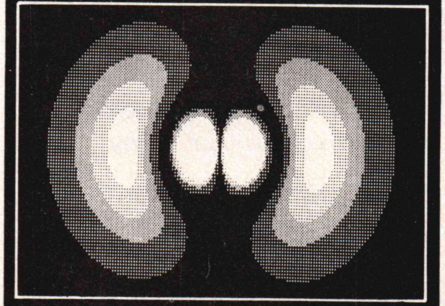
ISYS FORTH

for the Apple®] [

Fixed point speed can rival that of floating point hardware. But the details have been a well kept secret—until now. The following graphs were generated by fixed point examples from the ISYS FORTH manual.



Parallel Resonance with Damping
BASIC 213 sec ISYS FORTH 27 sec



Hydrogen 3p Orbital Cross-section
BASIC 492 sec ISYS FORTH 39 sec

- **Fast** native code compilation. Sieve benchmark: 33 sec
- **Floating Point**—single precision with trancendentals
- **Graphics**—turtle & cartesian with 70-column character set
- **Double Precision** including D*/
- **DOS 3.3 Files** read & written
- **FORTH-83** with standard blocks
- **Full-Screen Editor**
- **Formatter** for word processing
- **Macro Assembler**
- **Price: $99,** no extra charges

ILLYES SYSTEMS
PO Box 2516, Sta A
Champaign, IL 61820

Technical Information:
217/359-6039, mornings

For any Apple] [model, 48K or larger. Apple is a registered trademark of Apple Computer.

Circle **no. 48** on reader service card.

 (Listing Continued, text begins on page 102)

Listing One

```
        db      cr,lf
err2len equ     (this byte)-(offset err2)

buffer  equ     this byte       ; data is read here from
                                ; standard input

cseg    ends

        end     tee
```

End Listing One

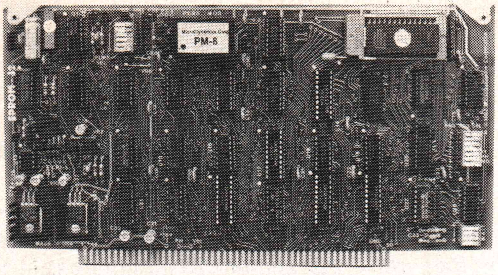
Listing Two

```
Programming Tip:   Jump Tables are a convenient,  efficient,
and general way of directing program flow based on a
multiple choice. Here is an implementation of a Jump Table
for the 68000.

*
*  Register DØ = choice index from Ø to N
*
start       equ         *
            add.w       dØ,dØ
            move.w      jtable(pc,dØ.w),dØ
            jmp         jtable(pc,dØ.w)
jtable      equ         *
            data        /jumpØØØ-jtable
            data        /jumpØØ1-jtable
            data        /jumpØØ2-jtable
            data        /jumpØØ3-jtable
            (etc)
            (etc)
            (etc)
jumpØØØ     equ         *
            (code)
            bra         continu
jumpØØ1     equ         *
            (code)
            bra         continu
jumpØØ2     equ         *
            (code)
            bra         continu
jumpØØ3     equ         *
            (code)
            bra         continu
            (etc)
            (etc)
continu     equ         *
            (code)


End of Listing 2
```

End Listings

S-100 EPROM PROGRAMMER

EPROM-32



- Field-proven board meets IEEE-696 standard.
- Programs 1K through 32K (byte) EPROMs.
- Textool zero-insertion-force programming socket.
- EPROM is programmed through I/O ports and can be verified through I/O ports *or* located in memory space for verification.
- Programming voltage generated on-board.
- Personality Modules adapt board to EPROMs:

PM-1—2508, 2758	PM-3—2732, 2732A	PM-6—68764
2516, 2716	PM-4—2564	PM-8—27128
PM-2—2532	PM-5—2764	PM-9—27256

- Feature-packed CP/M-compatible control software includes **fast programming algorithm.**
- One year warranty.

$269.95*
(A & T)

MicroDynamics Corporation

Suite 245 • 1355 Lynnfield Road • Memphis, TN 38119
(901)-682-4054

* Price includes EPROM-32, documentation and two Personality Modules(specify). Additional Modules—$7.95. Control software on 8" SSSD diskette—$29.95. UPS ground—$2.00, UPS air—$4.00, COD—$1.65, foreign add $15.00, VISA & MASTERCARD welcome.

See Dec. 1983 **Microsystems** for a review of the EPROM-32.

Circle **no. 39** on reader service card.

Fortran Scientific Subroutine Package

Contains Approx. 100 Fortran Subroutines Covering:

1. Matrix Storage and Operations
2. Correlation and Regression
3. Design Analysis
4. Discriminant Analysis
5. Factor Analysis
6. Eigen Analysis
7. Time Series
8. Nonparametric Statistics
9. Distribution Functions
10. Linear Analysis
11. Polynomial Solutions
12. Data Screening

Sources Included

$295.00

FORLIB-PLUS™

Contains three assembly coded LIBRARIES plus support, FORTRAN coded subroutines and DEMO programs.

The three LIBRARIES contain support for GRAPHICS, COMMUNICATION, and FILE HANDLING/DISK SUPPORT. An additional feature within the graphics library is the capability of one fortran program calling another and passing data to it. Within the communication library, there are routines which will permit interrupt driven, buffered data to be received. With this capability, 9600 BAUD communication is possible. The file handling library contains all the required software to be DOS 3.0 PATHNAME compatible.

STRINGS & THINGS™

Support for CHARACTER MANIPULATION (string support), SHELL, BATCH, MUSIC, CMD LINE, and ENVIRON CTRL.

$69.95 each

P.O. Box 2517
Cypress, CA 90630

ALPHA COMPUTER SERVICE **(714) 894-6808**

California residents, please add 6% sales tax.

Versions available for IBM Professional Fortran or MICROSOFT 3.2 Fortran

Circle **no. 1** on reader service card.

Indexed File Manager using B-Trees

$75.00
+ 2.00 Postage
source included

C Programmers, We provide the record handling that C left out.

The softfocus BTree library is a record oriented function package that uses balanced BTree indexing for guaranteed fast access. Add our functions to your C library and greatly reduce application development time.

- **High speed** keyed and sequential file handling. Up to 16.7 million records per file.
- Source code supplied; conforms to **K&R standard** to ensure portability.
- **No royalties** on application programs.
- Documentation and **example programs** included to help you use BTrees.
- **Full feature** product at a fraction of the cost of competing BTree software.

Join the growing number of satisfied programmers using softfocus BTrees.

To order call

softfocus

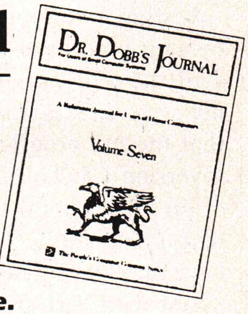
1277 Pallatine Drive
Oakville, Ontario L6H 1Z1
(416) 844-2610

Credit cards accepted. ***Dealer inquiries invited.***

Circle **no. 89** on reader service card.

Dr. Dobb's Journal

Bound Volumes



Every Issue Available For Your Personal Reference.

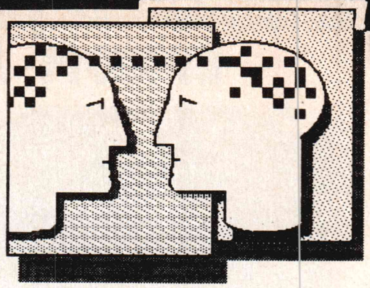
We are pleased to offer this special discounted price to **DDJ** readers who order directly from us. From the nostalgia of Volume One—with authors like Steve Wozniak, Dennis Allison, Sol Libes, and more—to the technical maturity of Volume Seven, **Dr. Dobb's Journal Bound Volumes** are the ideal addition to your reference collection. They contain many issues which are no longer available and you get twelve issues for the price of seven individual back issues!

Send $26.75 for volume 1, $27.75 each for volumes 2–6, $30.75 for volume 7, or $165 for all seven and SAVE!

Please add the following per book: $2.50 for UPS, $1.25 for U.S. Mail, or $3.25 for Foreign surface mail. Foreign Airmail rates available on request. Delivery times are one week for UPS or 6–10 weeks for U.S. or Foreign Mail.

Mail to: DDJ, 2464 Embarcadero Way, Palo Alto, CA 94303

Please allow 6–9 weeks for delivery.



by Robert Blum

Bill Wilson of Pullman, Washington wrote several months ago describing a program he had written to simplify batch processing. He also included a patch to enable PIP to accept null command lines from a submit file. I am sure you will find, as I did, that both are most useful and real time-savers.

The patch described here is not approved by DRI; therefore you should first backup your distribution version of PIP and retain it for use in the unlikely event that you experience a problem. I further suggest that you exercise extreme caution as you modify PIP and install it on your system. The installation steps listed here are straightforward and shouldn't take more than a few minutes to enter; but do be mindful not to overwrite anything of value.

Listing Two (page 120) is the step by step procedure that I used to install the null line patch into PIP on my system. The first step is to verify that the PIP program being modified is version 1.5. This is the only version of PIP that I know the patch to work with successfully. To verify that you have the right version of PIP load PIP.COM with SID or DDT and display memory between 200h and 240h. At 0233h should begin the version number, which must be 1.5. Next, carefully enter the series of patches listed. And finally, save the modified program back onto disk under a new name ready for testing.

CP/M Version 2.2 Improved Batch Procedures

Batch mode automatic sequential command processing under CP/M 2.2 using SUBMIT is very inconvenient. Especially when using XSUB simply to pass a few parameters to a program. For instance, to copy a few files from one disk to another requires that ED or some other text editor first be used to build a disk file of the desired control statements. Then SUBMIT must be run to build yet another disk file of the very same control statements but in a format suitable for the CCP to use. The alternative approach is to enter each control statement from the keyboard as the copy program asks for them. Which is fine providing not too many files are being copied or you have plenty of time.

The prospect of continually being hampered by a lengthy process when using batch procedures prompted me to develop a stand-alone $$$.SUB file generator called BATCH. A source listing for BATCH written for the Z80 is given in Listing One (page 114).

The BATCH program served another very useful purpose as well. I was in the process of developing a CCP replacement in conjunction with a Computer Science student at WSU that was to have a built-in SUBMIT file generator. Thus it was important to ensure that the $$$.SUB file processor within the CCP replacement worked properly. Program BATCH was developed for this purpose, as well. As presented here it may be used to replace the combination of ED and SUBMIT for simple BATCH mode command processing.

The null line required for exiting from PIP, a line containing a CR only, also caused us a problem in the development of the CCP replacement. It is not difficult for a program like BATCH to produce a null line in a command string, but the processing of a null line by the $$$.SUB file processor caused problems. That is, it is most difficult for the $$$.SUB file processor to distinguish between a null line intended for exiting from PIP and one to designate that the end of the $$$.SUB has been found without a lot of code that uses an excessive amount of space.

A $$$.SUB file has a rather tricky construction with one command per sector stacked in reverse order. The $$$.SUB processor opens the $$$.SUB file, goes to the end of the file, reads in the last sector, decrements the record count in the $$$.SUB File Control Block (FCB), closes the $$$.SUB file, and then executes the command. In other words, the $$$.SUB file processor peels commands one at a time from the end of a $$$.SUB file. The crux of the PIP null line problem is that the $$$.SUB processor quits and erases the $$$.SUB file at the end of the file or if the character count is zero.

In order to make XSUB, PIP, and the CCP replacement function efficiently I decided to take a stab at modifying PIP. I spent an evening at home disassemling PIP on my S-100 buss Z-80 based system running under CP/M 2.2. I have a very good disassembler so the process was not very difficult. I was quite amazed by the old extra code for paper tape handling that is unused but still retained within PIP. Needless to say, I was not very impressed by what I found within PIP. However, because of the unused code, it was simple to insert some new code to solve the null line problem. Extensive testing with BATCH, XSUB, PIP, and the CCP replacement indicated that the fix worked fine. PIP would now exit on a null line or single character line. The modified version of PIP would only attempt to process an input line with two or more characters.

The next morning while shaving it dawned on me that an even simpler fix to the PIP null line problem was possi-

ble. Since it was Saturday and I did not have to go to work, I gave my new fix a try with the same results as before.

The procedure for installing the simple PIP fix using SID is shown here in Listing Two. PIP uses the buffered line function for input and does a CPI 00 at location 054Fh to check the number of characters entered. The JNZ at 0551h causes PIP to exit if and only if the character count is zero. That is, a null line is required to exit PIP. A simple fix involves changing the CPI 00 to a SUI 02 and the JNZ to a JP. PIP will now exit if the character count is less than 2. This modification will allow XSUB, PIP, CCP, etc., to all work in harmony.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 200**.

CP/M Exchange (Text begins on page 112)

Listing One

```
;
; ** BATCH version 2.1**
;     A program to generate a $$$.SUB submit file
;      for immediate processing under CP/M 2.2 .
;
;--NOTES:
;         1) The $$$.SUB file is created on disk unit A
;            since the D. R. utility XSUB will only operate
;            on disk unit A. If BATCH is run from disk B,
;            for example, the $$$.SUB file will be created
;            on disk unit A and then processed from disk A.
;            All files referenced in the submit file created
;            by BATCH must be on the disk unit from which
;            BATCH is being run including PIP and XSUB.
;
;         2) Use DDT to Modify PIP as follows to solve the
;            null line PIP exit problem: change 053D from
;            2A to 23, 0545 from FE to D6, 0550 from 00 to
;            02, and 0551 from C2 to F2. Exit DDT by entering
;            G0, and then do a SAVE 32 PIP.COM.  The modified
;            version of PIP outputs # as an input request
;            symbol and will exit on a null or single
;            character input line.
;
;         3) Input line must not exceed input buffer length
;            which is 128 characters.
;
;         4) Example use of BATCH:
;               >BATCH<RET>
;               ->XSUB/PIP/B:=A:XDIR.COM/X/DIR A:/DIR B:<RET>
;
; ** PROGRAM DEFINITIONS **
CR      EQU     0DH             ;CARRIAGE RETURN
LF      EQU     0AH             ;LINE FEED
BDOS    EQU     005H            ;ADDR CP/M BDOS FUNCTION CALLS
;
        ORG     100H            ;PROG ORG
;
START:  LD      HL,00H
        ADD     HL,SP           ;GET CP/M STACK POINTER
        LD      (CPMSTK),HL     ;SAVE IT
        LD      SP,STACK        ;SET UP NEW STACK
        LD      DE,PROMPT       ;PTR TO SIGNON MSG
        CALL    PUTSTR          ;SEND STRING
        CALL    SETDMA          ;SET DKS BUF ADDR
        CALL    GETBUF          ;GET INPUT
        LD      HL,INPBUF+1     ;PTR NUM INP CHRS
        LD      A,(HL)          ;GET NUM INP CHRS
        LD      C,A             ;SAVE CHR CNT IN C
        DEC     A               ;CK IF ANY CHRS IN BUFF
        JP      M,BCKCPM        ;NO CHRS, BACK TO CP/M
        JP      Z,BCKCPM        ;NO CHRS, BACK TO CP/M
        LD      A,0FFH          ;GET FF=START OF INPUT
        LD      (HL),A          ;SET CHR CNT=FF
        LD      A,C             ;GET CHR CNT AGAIN
        INC     A               ;ADD ONE FOR 01 END CHR
;--ADD A TO HL TO FORM ADDR END INP STRING--
        CALL    ADATHL
        LD      (HL),01         ;INDICATE END OF INPUT
;
;--SELECT DISK UNIT A--
```

(Continued on page 116)

THE SIMPLE APPROACH IS THE SYMBOL APPROACH.

```
10 S=0
20 FOR I=1 TO 100
30 INPUT X
40 IF X = 0 GOTO 70
50 S=S+X
60 NEXT I
70 PRINT S/(I-1)
```

BASIC
A program to calculate averages...

```
   REAL X(100)
   READ*,N,(X(I),I=1,N)
   S=0
   DO 10 I=1,N
10 S=S+X(I)
   PRINT *,S/N
   END
```

FORTRAN
just shrunk from seven lines...

```
(+/X)÷ρX←⎕
```

POCKET APL
to one.

INTRODUCING POCKET APL™.

Pocket APL, a new PLUS★WARE™ product, symbolizes a whole new way to solve problems. Faster than Fortran. Simpler than Basic. And at a cost much less than Cobol and many other programming languages. Its use of symbols makes it concise and efficient— powerful and productive.

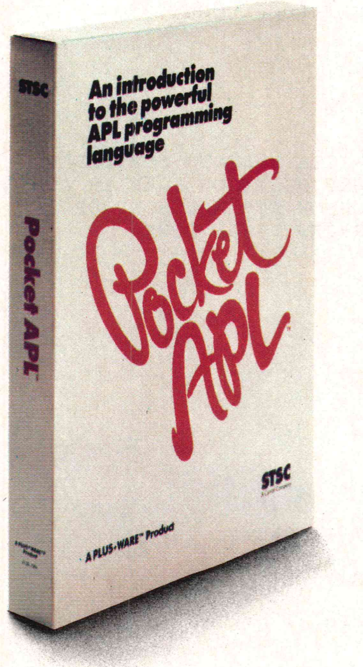
WORKING IN SHORTHAND = WORKING FASTER, SMARTER.

Pocket APL allows you to shrink the length of your programs. Because just a few symbols say what takes lines and lines to say in other programming languages. So Pocket APL cuts the drudgery and need for tedious subroutines and long lists of commands.

GET FLEXIBILITY > WITH CANNED SOFTWARE.

Pocket APL is a complete APL implementation with enhancements like online HELP, windowing, report formatting, dual file system, and debugging aids. It's also a powerful online calculator. So you don't have to switch back and forth between programs or from your hand-held calculator to the computer.

And the symbols? Simple. You'll learn them fast. They'll become as second nature to you as +, −, ×, and ÷. Once you start using them, you'll be programming four to 10 times faster than with conventional languages. And as your needs grow, you can easily upgrade to STSC's APL★PLUS®/PC System for even more features—like communications and graphics.



POCKET APL COSTS MUCH < YOU'D EXPECT.

Pocket APL makes programming easy. And priced at just $95, it's easy on the budget, too. It works with IBM PC's and compatibles and requires only 128K. So if problem-solving is taking up too much of your time, the answer is symbol. Pocket APL.

To order, or for more information, call **800-592-0050**. In Maryland, call **(301) 984-5123.**

Or write STSC, Inc., 2115 East Jefferson St., Rockville, MD 20852. All major credit cards accepted.

Problem-solving at the speed of thought.

STSC
A Contel Company

Pocket APL uses a soft character set for computers with IBM-compatible graphics board or color monitor; keywords for computers with monochrome. Optional character generating ROM can be ordered for IBM PC monochromes or Hercules monochrome boards.

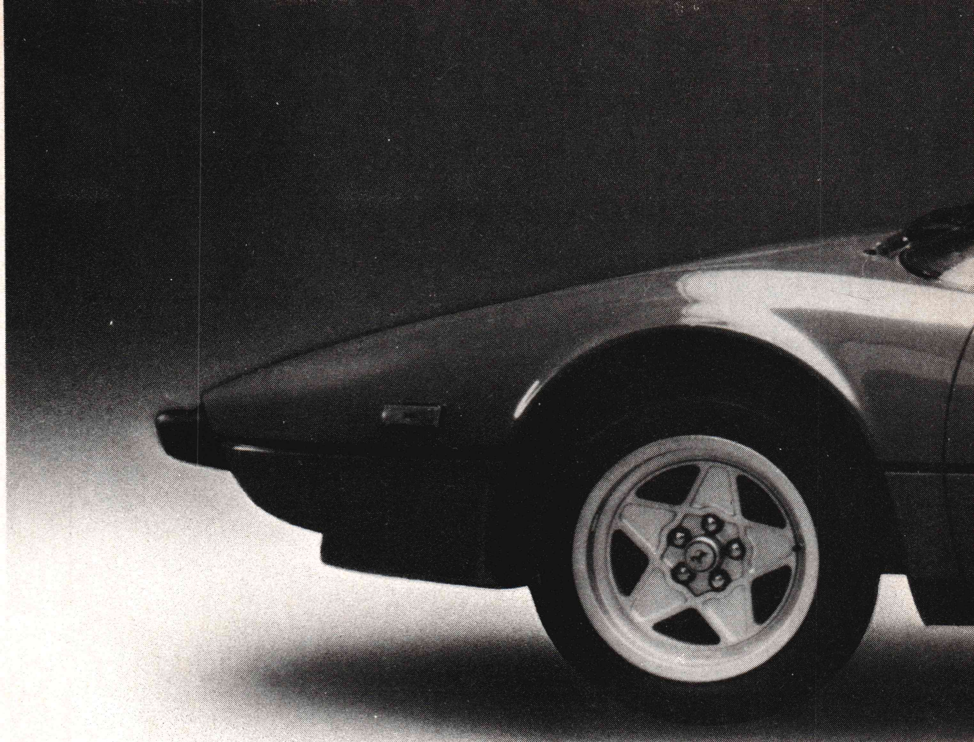
PLUS★WARE and Pocket APL are trademarks of STSC, Inc. APL★PLUS is a service mark and trademark of STSC, Inc., registered in the U.S. Patent and Trademark Office and in other countries.

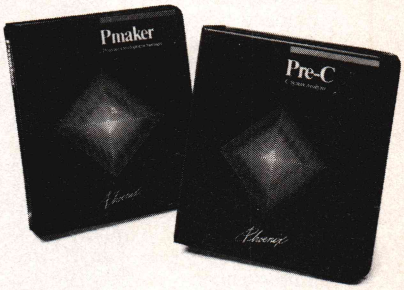
Circle **no. 110** on reader service

Listing One

```
        CALL    INIT
;
        CALL    DELETE          ;DELETE OLD $$$.SUB
        CALL    MAKE            ;MAKE NEW $$$.SUB
; SET CURRENT RECORD=0
        LD      HL,SUBFCB+32    ;HL=PTR CUR POS IN FCB
        XOR     A
        LD      (HL),A
;--CONVERT BUFFER TO UPPER CASE CHRS--
;--SKIP ANY LEADING BLANK CHARS IN 1ST STRING--
        CALL    CNVBUF
        LD      HL,INPBUF+2     ;HL=PTR INPUT BUFF
;--COUNT CHRS IN INPUT STRING AND SET HL TO END--
COUNT   CALL    CNTCHR
;--MAIN COMMAND SCANNING LOOP (RIGHT TO LEFT)--
SCNLOP  LD      A,(HL)          ;GET CHR
        CP      OFFH            ;CHECK IF DONE
        JR      Z,DONE          ;QUIT, DONE
        LD      C,0             ;BACKUP TO PREVIOUS /
SCNLP2  DEC     HL              ;DEC CHZ PTR
        LD      A,(HL)          ;GET CHR
        INC     C               ;INC COUNT
;--FF=BEGINNING (END) OF INPUT STRING--
        CP      OFFH            ;CHECK IF LAST CMND
        JR      Z,PUTCMN        ;PUT LAST COMMAND IN SUB FILE
;--/=BEGINNING (END) OF ONE COMMAND STRING--
        CP      '/'             ;CHECK FOR / BETWEEN COMMANDS
        JR      NZ,SCNLP2       ;LOOP TILL / OR FF
;--PUT COMMAND INTO SUBMIT FILE--
;--USE CP/M DEFAULT BUFFER AREA--
PUTCMN  PUSH    HL              ;SAVE POINTER
        DEC     C               ;DEC FOR / CHR
        INC     HL              ;SKIP OVER /
        LD      A,C
        LD      (80H),A         ;PUT LENGTH IN BUFFER
        LD      DE,81H
        CALL    BLKMOV          ;MOVE COMMAND TO BUFFER
        EX      DE,HL           ;SET HL TO END OF COMMAND
;--ZERO REST OF BUFFER--
        LD      A,127
        SUB     A,C
        LD      B,A             ;SET LOOP COUNT
        LD      C,0
ZROLOP  LD      (HL),C
        INC     HL
        DJNZ    ZROLOP
;--WRITE ONE SECTOR--
        CALL    WRITE           ;WRITE BUFF TO SUB FILE
        POP     HL              ;GET POSITION POINTER AGAIN
        INC     A               ;CHECK FOR WRITE ERROR
        JR      NZ,SCNLOP       ;REPEAT TILL DONE.
;--DISK ERROR--
        LD      DE,NSPMSG
        JP      SERMSG
;
;--DONE, BACK TO CP/M--
;--DO A SYS RESET TO START SUBMIT--
DONE:   CALL    CLOSE
        JP      000H

;
;--COUNT THE CHARS IN THE COMMAND--
;  MOVE HL TO END OF COMMAND
CNTCHR: LD      C,1             ;INIT CNT=1
CNTLOP: LD      A,(HL)
        CP      ODH             ;END IF CR
        RET     Z
        CP      01H             ;END IF CHR=01H
        RET     Z
        INC     C
        INC     HL
        JR      CNTLOP
;
;--ADD A TO HL--
```

(Continued on page 118)





Pmaker
Pre-C

CP/M Exchange (Listing Continued, text begins on page 112)

(Listing Continued, text begins on page 112)

Listing One

```
ADATHL  ADD     A,L
        LD      L,A
        RET     NC
        INC     H
        RET
;
;--BLOCK MOVE--
;   HL=SOURCE, DE=DESTINATION, C=NUMBER OF CHARS
BLKMOV  LD      B,00H           ;ZERO B
        LDIR
        RET
;
;--CONVERT BUFFER TO UPPER CASE--
CNVBUF  LD      HL,INPBUF+2
        LD      D,H
        LD      E,L
;--SKIP LEADING BLANK CHARS--
SKPLOP  LD      A,(HL)
        CP      ' '
        JR      NZ,CNVLOP
        INC     HL
        JR      SKPLOP
CNVLOP  CALL    GETCHR
        LD      (DE),A          ;MOVE CHR
        INC     DE
        INC     HL
        CP      01H             ;END ?
        JR      NZ,CNVLOP
        RET
;
;--GET NEXT CHAR AND TO CONV UPPER CASE--
GETCHR  LD      A,(HL)          ;GET CHR
        CP      07BH            ;ABOVE z ?
        RET     NC
        CP      061H            ;BELOW a ?
        RET     C
        AND     05FH            ;LOW TO UP
        RET
;
;--GET BUFFERED LINE OF COMMANDS--
GETBUF: LD      DE,INPBUF       ;INP BUFF ADDR
        LD      C,10
        CALL    BDOS
        RET
;
;--SET DISK BUFFER ADDRESS(DMA)--
SETDMA  LD      DE,080H         ;BUFF ADDR
        LD      C,26
        CALL    BDOS
        RET
;
;--WRITE CONTENTS OF BUFF TO SUB FILM--
WRITE:  LD      DE,SUBFCB
        LD      C,21
        CALL    BDOS
        RET
;
;--CLOSE $$$.SUB--
CLOSE   LD      C,16
        LD      DE,SUBFCB
        CALL    BDOS
        RET
;
;--MAKE NEW $$$.SUB FILE--
MAKE:   LD      DE,SUBFCB
        LD      C,22            ;MAKE
        CALL    BDOS
        INC     A               ;FF=NO SPACE
        JP      Z,NOSERR        ;NO SPACE ERROR
        RET
;
;--DELETE ANY OLD  $$$.SUB FILE--
DELETE: LD      DE,SUBFCB
        LD      C,19
```

(Continued on page 120)

Listing One

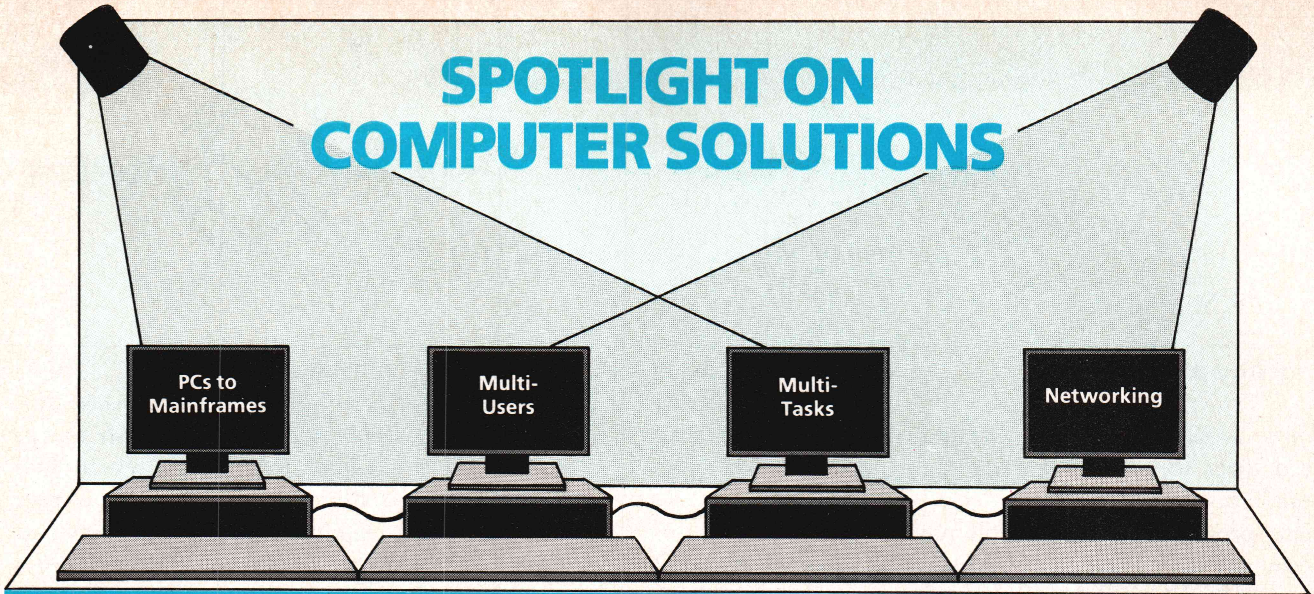
```
        CALL    BDOS
        RET
;
;--INIT AND SELECT DISK A--
INIT:   LD      C,13            ;RESET
        CALL    BDOS
        RET
;
;--SEND STRING TO TERMINAL--
; --DE = POINTER TO STRING ADDRESS--
PUTSTR  LD      C,9
        CALL    BDOS
        RET
;
;--MESSAGES--
PROMPT  DEFB    CR,LF,'BATCH v2.1 Submit File Generator'
        DEFB    CR,LF,'for use with XSUB under CP/M 2.2  .'
        DEFB    CR,LF,'Enter Command String and then <RET> .'
        DEFB    CR,LF,'Separate Each Command with a / Symbol .'
        DEFB    CR,LF,'/X may be used to exit modified version of PIP .
        DEFB    CR,LF,'->$'
NSPMSG  DEFB    CR,LF,'Disk Full--No More Space'
        DEFB    CR,LF,07H,07H,'$'
NDSMSG  DEFB    CR,LF,'Directory Full--No More Space'
        DEFB    CR,LF,07H,07H,'$'
;
NOSERR  LD      DE,NDSMSG
;--SEND ERROR MESSAGE AND BACK TO CP/M--
SERMSG  CALL    PUTSTR
BCKCPM  LD      HL,(CPMSTK)
        LD      SP,HL           ;RESET STACK POINTER
        RET                     ;BACK TO CP/M
;
;-- FILE CONTROL BLOCK--
SUBFCB  DEFB    OH              ;DRIVE CODE
        DEFB    '$$$     SUB'   ;FILE NAME
        DEFB    OH,OH,OH,OH
        DEFS    17
;--CP/M STACK STORAGE--
CPMSTK  DEFW    OOH
;-- INPUT BUFFER AREA--
INPBUF  DEFB    128             ;BUF LENGTH
BUFCNT  DEFB    OH              ;CHRS IN BUF
        DEFS    128
;--STACK AREA--
        DEFS    20              ;SIZE OF STACK
STACK   EQU     $               ;BOTTOM OF STACK
        END
```

End Listing One

Listing Two

```
B>SID PIP.COM
CP/M 3 SID - Version 3.0
NEXT MSZE  PC  END
1E00 1E00 0100 C8FF
#D200,240
0200 20 20 20 43 4F 50 59 52 49 47 48 54 20 28 43 29    COPYRIGHT (C)
0210 20 31 39 37 39 2C 20 44 49 47 49 54 41 4C 20 52  1979, DIGITAL R
0220 45 53 45 41 52 43 48 2C 20 20 50 49 50 20 56 45 ESEARCH,  PIP VE
0230 52 53 20 31 2E 35 03 01 06 01 00 24 24 24 20 20 RS 1.5.....$$$
0240 20
#S053D
053D 2A 23
053E CD .
#S054F
054F FE D6
0550 00 02
0551 C2 F2
0552 5E .
#WPIPNEW.COM,100,1E00
003Ah record(s) written.
#G0
```

End Listings



UNIX* SYSTEMS EXPO/85

Spring

BE THERE!
For Computer Solutions To

- Office/Factory Automation
- Engineering/Scientific Applications
- Text Processing
- Database Access

April 24–26
San Francisco Moscone Center

BE THERE!
To Learn About

- Linking PCs to MainFrames
- Multi-User/Tasking Operations
- Networking
- Clusters/Device Sharing

See exhibits from the leading and most innovative hardware and software companies. Attend the full conference program with over 40 user/marketing oriented sessions.

An exclusive production of
Computer Faire, Inc./
A Prentice-Hall Company
617/965-8350, 415/364-4294

Don't Miss UNIX Systems Expo/85 for:

- Engineers
- Scientists
- Corporate Users
- VARs
- Software Developers
- Vertical Marketers
- Systems Integrators
- Distributors
- Dealers
- Retailers

*UNIX is a trademark of AT&T Bell Laboratories

UNIX* SYSTEMS EXPO/85 *Spring*

Please send me information prior to April 1st on:

☐ **Exhibiting** ☐ **Attending**

Name ____________________

Title ____________________

Company ____________________

Address ____________________

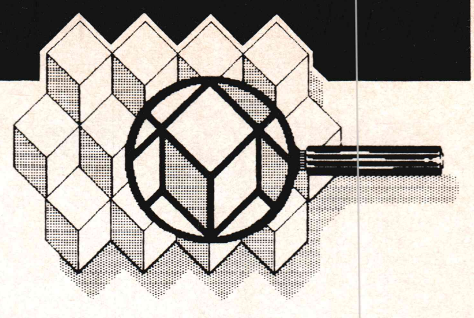
City/State/Zip ____________________

Phone ____________________

(DD)

Please send to: Computer Faire, Inc. 181 Wells Avenue Newton, MA 02159

Circle **no. 15** on reader service card.



by R. P. Sutherland

Hardcopy

Three articles in this issue illustrate techniques to generate various nonstandard printed characters. Despite their present utility, they will soon seem curious and quaint. The proliferation of the Macintosh and Mac-alike machines as well as the porting of Donald Knuth's TEX to micros (in tandem with the availability of laser printers) will result in easier ways of coaxing two and three dimensional graphic concepts into hardcopy. The Figure below shows a sample of output from TEX as well as a list of vendors who carry micro implementations.

Publications

Privacy Journal: An Independent Monthly on Privacy in the Computer Age contains the seeds of the antidote to *1984*. Subscriptions: $89.00 per year, $109.00 overseas. Contact Kathryn Ritter, *Privacy Journal*, P.O. Box 15300, Washington, D.C. 20003 (202) 547-2865. **Reader Service No. 101.**

Apple Access: User's Guide to Apple Computer-Related Periodical Literature for $19.95 is a reference for Apple literature. Contact Lee Webber, Stony Point Publications, Box 4467, Petaluma, CA 94953 (707) 778-8754. **Reader Service No. 103.**

The Datapro Complete Guide to Dial-Up Databases profiles more than a thousand accessible data bases in over 200 subjects. Contact Karen Morrell, Datapro Research Corpora-

"small" TEX implementations

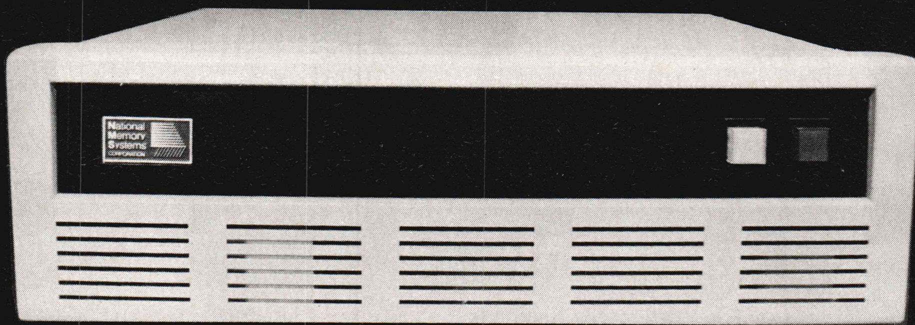
Computer	Processor	Contact	Organization, Address
Hewlett-Packard 3000	16-bit	Lance Carnes	TeXeT, 163 Linden Lane, Mill Valley, CA 94941; 415-388-8853
Hewlett-Packard 1000	16-bit	John Johnson	JDJ Wordware, Box 354, Cupertino, CA 95015; 415-965-3245
DEC PDP-11/44	16-bit		
Plexus, Onyx	Z8000	Dick Gauthier	TYX, 11250 Roger Bacon Dr., Suite 16, Reston, VA 22090; 703-471-0233
IBM PC	8086/88		
Apollo	MC68000	Thom Hickey	OCLC, Box 7777, Dublin OH 43017; 614-764-6075
		Bill Gropp	Dept. of Computer Science, Yale University, Box 2158, Yale Station, New Haven, CT 06520; 203-436-3761
		Pierre Clouthier	COS Information, 6272 Notre Dame West, Montreal, H4C 1V4, P.Q.; 514-935-4222
Hewlett-Packard 9836	MC68000	Jim Crumly	Hewlett-Packard, Box 15, Boise, ID 83707; 208-376-6000 x2869
Sun Microsystems	MC68000	Jim Sterken	Textset, Box 7993, Ann Arbor, MI 48107; 313-996-3566
		Rich Furuta	University of Washington, Computer Science, FR-35, Seattle, WA 98195; 206-543-7798
Cyb	MC68000	Norman Naugle	Mathematics Dept., Texas A&M University, College Station, TX 77843; 409-845-3104
Apple Macintosh, Lisa	MC68000	Barry Smith Dave Kellerman	Kellerman & Smith, 2343 SE 45th Av., Portland, OR 97215; 503-232-4799
Masscomp	MC68000	Bart Childs	Dept. of Computer Science, Texas A&M University, College Station, TX 77843; 409-845-5470
Synapse	MC68000	Dick Wallenstein	Comcon, 5 Underwood Ct., Delran, NJ 08075; 609-764-1720
PERQ/ICL		Jaap van 't Ooster	Océ, St. Urbanusweg 43, 5900 MA Venlo, Holland
IBM PC, XT, AT	8088, 80286	Lance Carnes	TeXeT, 163 Linden Lane, Mill Valley, CA 94941; 415-388-8853
IBM XT, AT	8088, 80286	David Fuchs	Dept. of Computer Science, Stanford University, Stanford, CA 94305
IBM XT, AT	8088, 80286	Ronny Bar-Gadda	446 College Av., Palo Alto, CA 94306; 415-326-1275

Figure

Try a mainframe tradition on your microcomputer today...

Desk Top Capacities

1000 mb
700 mb
335 mb
170 mb
85 mb



Rack Mount Capacities

1000 mb
960 mb
480 mb

The NMS PC 8000 Series
- 8-inch technology-based storage
- Access times of 15 milliseconds

...National Memory Systems provides

Desk top storage to 700 megabytes:

With enhanced features for 1985, the NMS PC 8000 stays ahead of any herd with compact 8-inch technology Winchester Disk Storage systems ... used in mainframe and minicomputer installations. NMS originated use of these techniques in micro applications more than 18 months ago. We set the standard for high-capacity, high-performance mass storage systems.

Compatibility:

IBM PC-AT, XT, standard PC, and all compatibles, as well as a version to run your Texas Instruments PC, too! Ask about our multifunction, single slot disk tape controller.

Ideal for networking and distributed processing:

You gain up to 5 times the speed of 5¼-inch technology disk systems. NMS Disk Systems will not impose limitations on your computer.

File-by-File Tape Back-up:

Add our stand-alone PC.25 60 mb File-by-File and streamer ¼-inch tape system, or ½-inch PC 9000 Series, ANSI-IBM format compatible 9-track system, and you have the total solution.

Laser Optical Disk:

Be revolutionary with the 1 gigabyte NMS PC 007, a non-erasable archival system that gives you 1000 megabytes of on-line data at 1/20th the cost of tape media.

Easy Upgrades:

Mix disk capacities, disk with tape, disk with Laser Disks to meet your needs. One single-slot controller does it all.

Call Us:

All products provided by NMS are protected with our standard 1-year warranty, and optional Nationwide field services program.

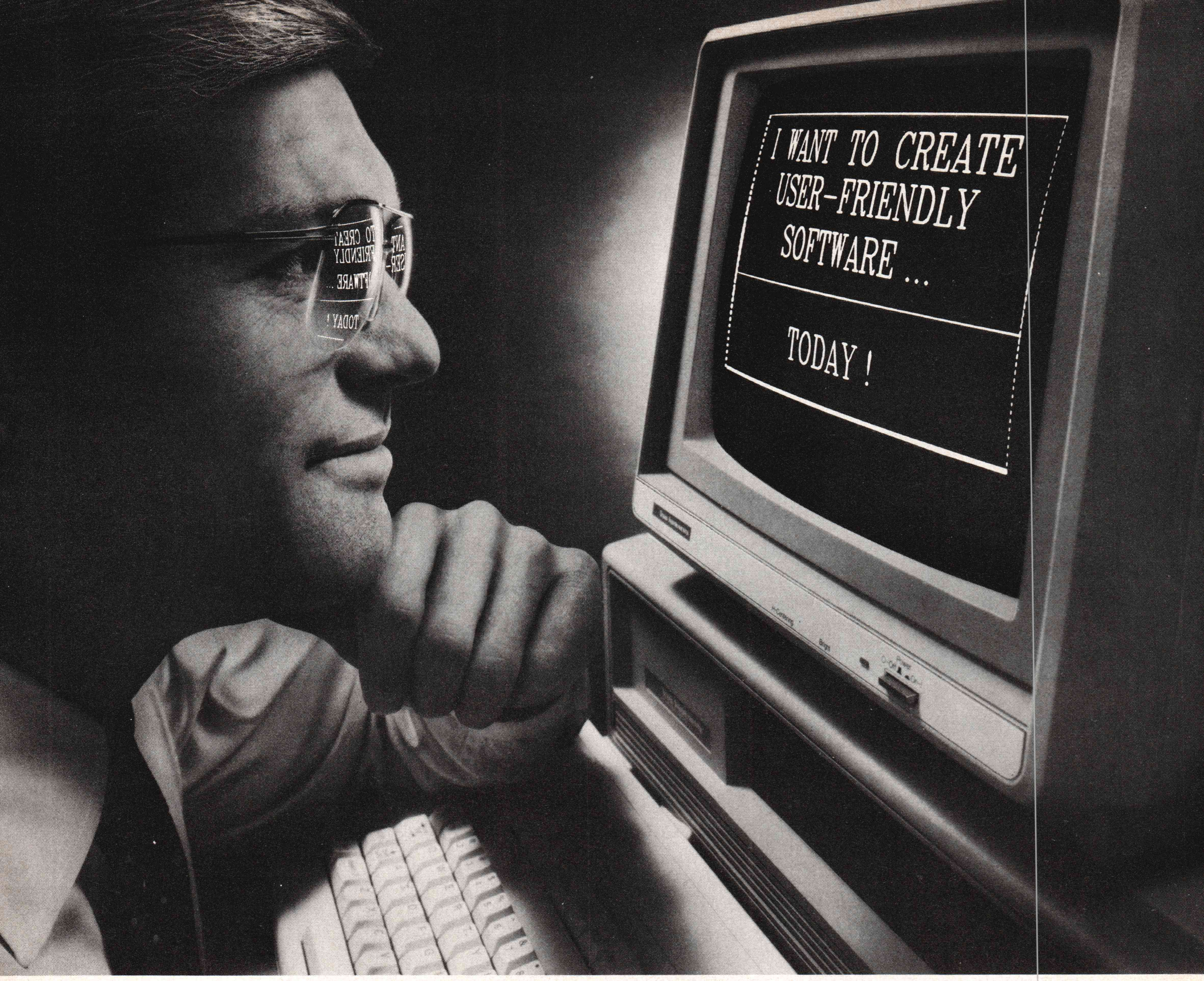
National Memory Systems CORPORATION

355 Earhart Way
Livermore, CA 94550
(415) 443-1669

Phone 415-443-1669

TWX 9103866006
TELEX 821892NMSUD

Circle **no. 87** on reader service card.



TI NaturalLink: user-friendly software with a developer-friendly tool kit.

TI makes it easier for developers to make software easier for PC users.

TI's innovative NaturalLink™ interface eliminates complex commands by translating them into the user's language: plain English. And the TI NaturalLink technology tool kit lets you develop interfaces for new or existing software in a way that speaks your language: simplicity. And better sales. The tool kit contains all the software and manuals you need to design, develop and test NaturalLink applications—quickly—on the TI Professional Computer. And our NaturalLink IBM® option contains the linkable object code you need to run your programs on the IBM PC, XT, AT or compatibles.

For more information or a demonstration, call us at 1-800-527-3500 or mail the coupon. We'll prove how developer-friendly we can be.

You're speaking my language, TI.

☐ Please mail me more information.

☐ Contact me to arrange a demonstration.

Name ______________________

Company ______________________

Address ______________________

City ______________________

State ____________ ZIP ____________

Mail to: Texas Instruments Incorporated
P.O. Box 809063, Dept. DCC063Y3514C
Dallas, TX 75380-9063

TEXAS INSTRUMENTS

Creating useful products and services for you.

2774-70

tion, 1805 Underwood Boulevard, Delran, NJ 08705 (800) 257-9406 (in New Jersey 609/764-0100). Price: $145.00. **Reader Service No. 105.**

Frank Gaude's **Z-System Newsletter** for ZCPR3 users announces a Z-system electronic bulletin board at (213) 670-9465. Echelon, Inc., 101 First Street, Los Altos, CA 94022. **Reader Service No. 107.**

MSX World: The Newsletter Dedicated to the World of MSX Computers is published by MSX World, 39 W. 32nd Street, Suite 800, New York, NY 10001. **Reader Service No. 109.**

C

A demonstration of the **Safe Runtime Analyzer** (Catalytix Corporation) is available at (617) 497-4829: 1200 baud, Unix System III, login is safec, password is catalytix. The demonstration allows one to upload C programs for analysis by the Safe C Runtime Analyzer, twenty-four hours a day. Contact Janice Tosi, Catalytix Corporation, 55 Wheeler Street, Cambridge MA 02138 (617) 497-2160. **Reader Service No. 111.**

The **C lib**, a C function library, provides over 200 routines including functions to convert floating point numbers from MicroSoft to 8087 NDP format and vice versa. Price: $145.00 from Vance Info Systems, 2818 Clay Street, San Francisco, CA 94115 (415) 922-6539. Contact: Carla Radosta. **Reader Service No. 113.**

A **mathematics library** compatible with most C compilers is available for $100.00 from Micro International, Box 47, Route 36, East Fairfield, VT 05448 (802) 827-3827. Contact: Bernard Hussels. **Reader Service No. 115.**

MSDOS

A special version of Microsoft's MSDOS 2.11 operating system is available for S-100 computers. **SB-86**, from Lifeboat Associates, requires a CompuPro System Support I card with G086 EPROM, and at least 64K of 24-bit addressable RAM. The installation routine supports both interrupt-driven and non-interrupt-driven console I/O. The price of SB-

86 is $275.00, available from Lifeboat Associates, 1651 Third Avenue, New York, NY 10128 (212) 860-0300. Contact: Doug Barth. **Reader Service No. 117.**

MasterForth Version 1.0 provides Forth-83 for the IBM PC family. MasterForth includes an 8088 macro-assembler and full interface to MSDOS 2.1. Price $125.00 from MicroMotion, 12077 Wilshire Boulevard #506, Los Angeles, CA 90025 (213) 821-4340. Contact: Linda Kahn. **Reader Service No. 119.**

MJ, from Mother Jones' Son's Software Corporation, is a utility that co-exists with MSDOS and permits one to open a window and reassign a key up to a thousand keystrokes. Also, MJ allows one to speed up the screen cursor in five increments and to expand the keystroke buffer from sixteen characters to over a thousand. MJ is not copy protected but if you abuse your backup privilege, you forfeit your soul to the author of the program. Price: $30.00 for object program, or $70.00 for object program and assembly language source code. Mother Jones' Son's Software Corporation, 6310 Caballaro Boulevard, Buena Park, CA 90620 (714) 522-7762. **Reader Service No. 121.**

Microcomputer owners who depreciate equipment for tax reasons are often approached by the IRS and asked to account for every minute spent on the computer. **TaxLog** is a product that keeps track of how much time one spends on various activities. TaxLog calculates the percentage of tax deductible use and prints a choice of reports that meet tax code requirements. Taxlog is part of Bellsoft's "Pop-Up" line of desk tools. TaxLog costs $39.95 from Bellsoft, 2820 Northrup Way, Bellevue, WA 98004 (206) 828-7282. Contact: Richard Leeds. **Reader Service No. 123.**

Macintosh

The **Public Domain Exchange** announces over a hundred public domain programs for the Macintosh. A listing of programs is available for $2.00. An introductory set of the three most popular disks and a catalog is $28.00. The Public Domain Exchange, 673 Hermitage Lane, San Jose, CA 95134 (408) 942-0309. Contact: Judy Rosenthal. **Reader Service No. 125.**

PortaAPL is now available for the Macintosh (512K). Price: $275.00. Portable Software, 60 Aberdeen Avenue, Cambridge, MA 02138 (617) 547-2918. Contact: Richard Smith. **Reader Service No. 127.**

Hard Disks

For a year now we have been trying to wear out a **Davong** 10 megabyte hard disk by driving our Davong Multilink system (our in-house LAN) 24 hours a day. So far it has been so reliable that we have become careless about backing anything up! Davong now has an 86-megabyte hard disk for the IBM PC. Davong's hard disk for the Macintosh is called **Mac Disk**. Mac Disks range from 10 to 43 megabytes and come with software that allows the user to configure and maintain multiple volumes. Contact John Houthton, Davong Systems, Inc., 217 Humboldt Court, Sunnyvale, CA 94089 (408) 734-4900. **Reader Service No. 129.**

Data Technology's **TeamMate** line of hard disks includes a Kodak-based 3.3 megabyte 5 1/4 inch floppy. I found the TeamMate hard disk easy to install and reliable. The Kodak floppy can be used to back up the hard disk or it can be used as an additional volume. The 3.3 megabyte floppy makes sense as a replacement to the AT's 1.2 megabyte drive because it more than doubles the capacity and it can write 48 TPI floppies more reliably than the AT's drive. Data Technology Corporation, 2775 Northwestern Parkway, Santa Clara, CA 95051 (408) 496-0434. Contact: Steve Roberts. **Reader Service No. 131.**

Single Board Computers

ForthCard, a single board computer on the STD bus with Forth kernel built in, features on-board EPROM programming and supports compilation of code from another computer. ForthCard is $499.00 in single quantities. An OEM ForthCard is also available. Contact: Ken Arnold, Hi-Tech Equipment Corporation, 9560 Black Mountain Road, San Diego, CA 92126 (619) 566-1892. **Reader Service No. 133.**

Motel Computers Ltd. has a 68000/Z80 based single board computer called **CYPHER**. Motel Computers Limited, 174 Betty Ann Drive, Willowdale, Ontario M2N 1X6 (416) 221-2340. Contact: Victor Pierobon. **Reader Service No. 135.**

Sundries

Maynard Electronics, Inc., offers a board that enables an IBM PC to run programs in DOS 3.0 that take advantage of extended memory. There are no software drivers required and the company claims the system runs as fast as the AT. Maynard Electronics, 430 Semoran Boulevard, Suite 207, Casselberry, FL 32707 (305) 331-6402. Contact: David Knapp. **Reader ServiceNo. 141.**

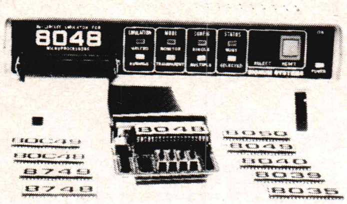
The **Public Domain Users Group** has expanded to include Commodore 64 users. For information and listings of the library disks, send SASE to Public Domain Users Group, P.O. Box 1442, Orange Park, FL 32067. **Reader Service No. 137.**

CommuniTree Group's **First Edition** (for IBM PC) is a program that lets people create dial-up information and conferencing services. First Edition can be used on PBX and PABX systems. The San Francisco CommuniTree Group provides an on-line newsletter and update service. "The FIG Tree," in Hayward, California, is a network for Forth programmers (415) 538-3580. First Edition is priced at $250.00. CommuniTree Group, 1150 Bryant Street, San Francisco, CA 94103 (415) 861-TREE. **Reader Service No. 139.**

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service No. 201.

IN-CIRCUIT EMULATOR
For 8050, 8049 and 8048 μC's



For Hardware/Software Design the E232-48 replaces the target microcomputer for complete emulation under control of the host computer. It emulates all of the above μC's plus their ROM-less and CMOS versions. It's features are:

★ Real time emulation up to 11 Mhz.
★ Full 4K Emulation Memory.
★ Hardware Breakpoints.
★ In-line Assembler and Disassembler.
★ Upload, Download & Terminal mode drivers for IBM-PC, CP/M-80 and CP/M-86 are included.

Cross Assemblers for 8048 series and other μP's running on the above host systems-From $150
E232-48 in-circuit emulator:....................... $1795

SIGNUM SYSTEMS
726 Santa Monica Blvd.
Santa Monica, CA 90401 (213) 451-5382

Circle **no. 106** on reader service card.

Now With Windowing!
$49.95 Basic Compiler

MTBASIC

Features:

Multitasking	Windowing
Handles interrupts	Interactive
Fast native code	Compiles quickly
Floating point	No runtime fee

MTBASIC is a true native code compiler. It runs *Bytes*'s Sept. '81 seive in 26 seconds; interpreters take over 1400 seconds! Because MTBASIC is multitasking, it can run up to 10 Basic routines at the same time, while displaying ten separate windows. Pop-up/down menus are a snap to implement.

MTBASIC combines the best of interpreters and compilers. To the programmer, MTBASIC appears to be an extremely fast interpreter. MTBASIC compiles a typical 100 line Basic program in 1 second, yet it generates blindingly fast code. No more waiting for long compiles.

AVAILABLE for CP/M (Z-80) and PC-DOS systems.

ORDERING: Specify format when ordering. We accept Visa, MC, checks and COD. Send $49.95 plus $3.50 shipping and handling ($10 overseas) to:

SOFTAID, Inc.
P.O. Box 2412 Columbia, MD 21045-1412
301/792-8096

Circle **no. 88** on reader service card.

C Users' Group

Over 40 volumes of public domain software including:

- compilers
- editors
- text formatters
- communications packages
- many UNIX-like tools

Write or call for more details

The C Users' Group
415 E. Euclid • Box 97
McPherson, KS 67460
(316) 241-1065

Circle **no. 17** on reader service card.

A general purpose programming language for string and list processing and all forms of non-numerical computation.

SNOBOL4+ —the entire SNOBOL4 language with its superb pattern-matching facilities • Strings over 32,000 bytes in length • Integer and floating point using 8087 or supplied emulator • ASCII, binary, sequential, and random-access I/O • Assembly Language interface • Compile new code during program execution • Create SAVE files • Program and data space up to 300K bytes RAM

With **ELIZA** & over 100 sample programs and functions

Have you tried SNOBOL4+?

MUSIC ANALYSIS • TEXT PROCESSING • SYMBOLIC MATHEMATICS • ARTIFICIAL INTELLIGENCE
COMPILER PROTOTYPING • CRYPTOGRAPHY • LINGUISTICS • LIST PROCESSING • GAMES •

For all 8086/88PC/MS-DO 128K minimum 5¼ DSDD.

Send check, VISA, M/C To:

Prentice-Hall, Inc.
B&P Div. Attn. S. Dragin Englewood Cliffs, NJ.
800-624-0023 outside New Jersey

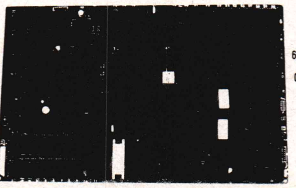
Circle **no. 20** on reader service card.

FOR THE **BEST** OF US . . .

THE CYPHER™

A COMPLETE 68000 & Z 80 SINGLE BOARD COMPUTER SYSTEM WITH ULTRA-HIGH-RES GRAPHICS!!

FREE 68000 FORTH AND CYPHER-DOS



FREE 68000 FORTH AND CYPHER-DOS

- 68000 & Z80 DUAL PROCESSORS (BEST OF BOTH WORLDS!).
- 256K to 1 MEGABYTE MEMORY. (4164 OR 41256 DRAM).
- DOUBLE DENSITY FLOPPY DISK CONTROLLER (8", 5¼" OR 3½" WD 2793).
- DMA CONTROLLER FOR FAST IMAGE TRANSFERS TO/FROM VIDEO MEMORY. (INT 8237).
- 2 RS232 SERIAL PORTS (ZSIO).
- 24 BIT ADDRESS MANAGEMENT FOR Z80.
- 4 LAYER P.C.B. (9¼" x 14¾").
- RUNS CP-M-80 2.2, CP-M-80 3.0, CP-M-68K
- ULTRA HIGH RESOLUTION GRAPHICS, 128K, PROGRAMMABLE UP TO 1024 x 1024 RESOLUTION (NEC 7220, GREAT FOR CAD SYSTEMS!).
- REAL TIME CLOCK (MULTITASKING CAPABILITY!).
- TWO CHANNELS OF D/A AND A/D, 12 BIT RESOLUTION (MUSIC! ROBOTICS! LAB WORK!).
- 16K TO 64K MONITOR EPROM.
- 4K TO 64K STATIC RAM.
- PROGRAMMABLE BAUD RATE GENERATOR.
- PARALLEL ASCII KEYBOARD INPUT.
- FULL 68000 EXPANSION BUS (60 PIN HEADER, BUFFERED).

•MANUAL .... $ 20.00
•BARE BOARD, EPROMS, Z80 BIOS, 68000 BIOS, Z80 MONITOR, 68000 MONITOR AND UTILITIES .... $ 399.95
•MINIMUM OPTIONED "CYPHER" WITH 68000/Z80, SERIAL I/O, 128K DRAM, 4K S RAM AND DISK CONTROLLER, ASSEMBLED AND TESTED .... $ 1,099.95
PARTS KIT .... $ 999.95
•COMPLETE "CYPHER" WITH 256K DRAM, 128K VIDEO DRAM, NEG 7220, REAL TIME CLOCK, A/D D/A DISC CONTROLLER, SERIAL I/O, ASSEMBLED AND TESTED $1,499.95
PARTS KIT $1,379.95
•KEYBOARD .... $ 94.95
•SWITCHING POWER SUPPLY .... $ 154.95
•CASE .... $ 94.95
•HARD DISC INTERFACE PLUG-IN CARD .... $ 125.00

VISA · master charge

ALL PRICES ARE IN U.S. DOLLARS
SHIPPING: 8% SHIPPING. EXCESS WILL BE REFUNDED
DELIVERY WILL BE MADE APPROXIMATELY 3 TO 5 WEEKS AFTER WE RECEIVE YOUR ORDER. VISA, MC, CASH
PRICES SUBJECT TO CHANGE WITHOUT NOTICE
CP-M IS A TRADEMARK OF DIGITAL RESEARCH

MOTOROLA · INTEL

MOTEL COMPUTERS LIMITED
174 BETTY ANN DRIVE, WILLOWDALE, TORONTO, ONTARIO, CANADA M2N 1X6
(416) 221-2340

Circle **no. 56** on reader service card.

BYSO™ LISP

has features that will delight both beginners and advanced programmers. A fast, reliable and complete interpreter for the IBM PC and true compatibles. $125 includes 100 pg. ref. manual and application notes that put you months ahead on useful projects *(making a hybrid language with C, accessing system functions and I/O ports, building your own dialect, etc.)*.

LEVIEN INSTRUMENT CO.
P.O. Box 31B
McDowell, VA 24458
703-396-3345

IBM PC is a trademark of the IBM Corp.

Circle **no. 25** on reader service card.

No source code for your REL files?

REL/MAC

converts a REL file in the Microsoft™ M80 format to a ZILOG™ or 8080 source code MAC file with insertion of all public and external symbols.

- **REL/MOD** lists library modules
- **REL/VUE** displays the bit stream
- 50 page manual with examples
- free brochure available
- **REL/PAK** includes all of the above

REL/PAK for 8080 only $99.95
REL/PAK for Z80 & 8080 .... $134.95
on 8"SSSD disk for CP/M™ 2.2

Send check, VISA, MC or C.O.D. to

MICROSMITH COMPUTER TECHNOLOGY
P.O. BOX 1473, ELKHART, IN 46515
1-800-622-4070
(Illinois only 1-800-942-7317)

Circle **no. 41** on reader service card.

ICs PROMPT DELIVERY!!!
SAME DAY SHIPPING (USUALLY)

OUTSIDE OKLAHOMA: NO SALES TAX

INCREASE XT MOTHERBOARD TO 640 Kbyte CONSUME NO EXPANSION SLOTS! $181.80

QUANTITY ONE PRICES SHOWN

DYNAMIC RAM			
256K	256Kx1	150 ns	$ 7.15
128K	128Kx1	150 ns	15.67
64K	64Kx1	150 ns	1.79
64K	64Kx1	200 ns	1.97
EPROM			
27256	32Kx8	250 ns	$26.97
27128	16Kx8	250 ns	9.97
27C64	8Kx8	200 ns	8.75
2764	8Kx8	250 ns	4.29
2732A	4Kx8	250 ns	4.87
2716	2Kx8	450 ns	3.21
STATIC RAM			
6264LP-15	8Kx8	150 ns	$10.70
6116LP-3	2Kx8	150 ns	3.43

OPEN 6½ DAYS: WE CAN SHIP VIA FED-EX ON SAT.

MasterCard/VISA or UPS CASH COD
Factory New, Prime Parts μP∞
MICROPROCESSORS UNLIMITED
24,000 South Peoria Ave.
BEGGS, OK. 74421
(918) 267-4961

Prices shown above are for March 11, 1985

Please call for current prices. Prices subject to change. Please expect higher or lower prices on some parts due to supply & demand and our changing costs. Shipping & insurance extra. Cash discount prices shown. Small orders received by 6 PM CST can usually be delivered to you **by the next morning, via Federal Express Standard Air @ $6.50!**

Circle **no. 64** on reader service card.

68000 Cross Assembler
Motorola VERSAdos + Compatible

Assembler, Linker, Object and Macro Librarian. Absolute and Relocatable Code, Macros, Includes, and Conditional Assembly. Structured Programming. No limit on source file size.

Unix (C) Compatible Source
$700

CP/M-80*	PC/DOS†	CP/M-86*
$200	$250	$250

Manual: $20
(refundable)

farbware

1329 Gregory
Wilmette, IL 60091

(312) 251-5310
after 5 p.m.

*Digital Research trademark. †IBM trademark. + Motorola trademark.

Circle **no. 57** on reader service card.

ADVERTISER INDEX

Advertising Sales Offices

East Coast
Walter Andrzejewski (415) 424-0600

Midwest/West Central
Michele Beaty (317) 875-0557

Northern California/Northwest
Shawn Horst (415) 424-0600

Southern California/Southwest
Beth Dudas (714) 643-9439

Classified Advertising
Lisa Boudreau (415) 424-0600

Advertising Director
Stephen Friedman (415) 424-0600

Advertising Coordinator
Lisa Boudreau (415) 424-0600

Assistant Advertising Coordinator
Jay Horvath (415) 424-0600

April 1985, #102

Dr. Dobb's Journal

Reader Service Card

A. Your primary job function: (Check one only)

- ☐ Company management (Pres., V.P., Treas., Owner, Gen. Mgr., Mktg. Dir.)
- ☐ Computer systems management (V.P. EDP., MIS Director, Data Processing Mgr., Data Communications Mgr., Network Planner)
- ☐ Engineering management (V.P. Engrg., Chief Engr., Tech. Director, Dir. R&D)
- ☐ Systems integrators (Systems Designer, Project Engr., Systems Application Engr., Technical Staff Members)
- ☐ Consultants (Computer/EDP/Data Communications Consultant)
- ☐ Educators (Educational users and instructors of computer technology)
- ☐ Systems/Programming specialists—mini-micro systems
- ☐ Other (Please specify)____________

B. Which languages are you MOST interested in?

☐ BASIC	☐ C	☐ PL/1
☐ Fortran	☐ LISP	☐ APL
☐ COBOL	☐ Prolog	☐ Logo
☐ Pascal	☐ Ada	☐ Smalltalk
☐ Modula-2	☐ Forth	Other ______

C. What is the operating system?

- ☐ CP/M (or derived)
- ☐ UNIX (or derived)
- ☐ MS-DOS (or derived)
- ☐ Other

D. Please indicate which of the following microcomputers you currently own and/or plan to buy in the next 12 months.

	Own	Plan to Buy
Apple	☐	☐
Commodore	☐	☐
Digital Equipment/DEC	☐	☐
Heath/Zenith	☐	☐
Hewlett-Packard	☐	☐
IBM	☐	☐
Macintosh	☐	☐
Radio Shack/Tandy TRS 80	☐	☐
Texas Instruments	☐	☐
Other (Specify)	☐	☐
None	☐	☐

E. What best describes the work you do with this microcomputer?

- ☐ Business functions
- ☐ Software/Hardware development
- ☐ Scientific/Engineering/R&D applications

F. Are you the decision maker or very influential in computer-related purchases at work?

☐ Yes ☐ No

G. Is your company a dealer, distributor, or systems house for microcomputers?

☐ Yes ☐ No

H. Subscriber

☐ Yes ☐ No

To obtain information about products or services mentioned in this issue, circle the appropriate number listed below. Use bottom row to vote for best article in issue. One card per person.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 25 26 27
28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108
109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135

Articles: 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214

This offer good until July 31, 1985

Name ____________________ **Phone (____)**__________

Address ____________________

City/State/Zip ____________________

April 1985, #102

Dr. Dobb's Journal

Reader Service Card

A. Your primary job function: (Check one only)

- ☐ Company management (Pres., V.P., Treas., Owner, Gen. Mgr., Mktg. Dir.)
- ☐ Computer systems management (V.P. EDP., MIS Director, Data Processing Mgr., Data Communications Mgr., Network Planner)
- ☐ Engineering management (V.P. Engrg., Chief Engr., Tech. Director, Dir. R&D)
- ☐ Systems integrators (Systems Designer, Project Engr., Systems Application Engr., Technical Staff Members)
- ☐ Consultants (Computer/EDP/Data Communications Consultant)
- ☐ Educators (Educational users and instructors of computer technology)
- ☐ Systems/Programming specialists—mini-micro systems
- ☐ Other (Please specify)____________

B. Which languages are you MOST interested in?

☐ BASIC	☐ C	☐ PL/1
☐ Fortran	☐ LISP	☐ APL
☐ COBOL	☐ Prolog	☐ Logo
☐ Pascal	☐ Ada	☐ Smalltalk
☐ Modula-2	☐ Forth	Other ______

C. What is the operating system?

- ☐ CP/M (or derived)
- ☐ UNIX (or derived)
- ☐ MS-DOS (or derived)
- ☐ Other

D. Please indicate which of the following microcomputers you currently own and/or plan to buy in the next 12 months.

	Own	Plan to Buy
Apple	☐	☐
Commodore	☐	☐
Digital Equipment/DEC	☐	☐
Heath/Zenith	☐	☐
Hewlett-Packard	☐	☐
IBM	☐	☐
Macintosh	☐	☐
Radio Shack/Tandy TRS 80	☐	☐
Texas Instruments	☐	☐
Other (Specify)	☐	☐
None	☐	☐

E. What best describes the work you do with this microcomputer?

- ☐ Business functions
- ☐ Software/Hardware development
- ☐ Scientific/Engineering/R&D applications

F. Are you the decision maker or very influential in computer-related purchases at work?

☐ Yes ☐ No

G. Is your company a dealer, distributor, or systems house for microcomputers?

☐ Yes ☐ No

H. Subscriber

☐ Yes ☐ No

To obtain information about products or services mentioned in this issue, circle the appropriate number listed below. Use bottom row to vote for best article in issue. One card per person.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 25 26 27
28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108
109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135

Articles: 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214

This offer good until July 31, 1985

Name ____________________ **Phone (____)**__________

Address ____________________

City/State/Zip ____________________

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #27346, PHILADELPHIA, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS
Dr. Dobb's Journal

P.O. BOX 13851
PHILADELPHIA, PA 19101

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #27346, PHILADELPHIA, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS
Dr. Dobb's Journal

P.O. BOX 13851
PHILADELPHIA, PA 19101